(12)

LEVEL

System Development Corporation

RESEARCH ON DEDUCTIVE INFERENCE FOR LARGE DATA BASES

FINAL TECHNICAL REPORT

Covering the Period

1 April 1976 through 30 December 1978

CHARLES KELLOGG AND IRIS KAMENY

31 JANUARY 1979

Prepared for:

Office of Naval Research, Arlington, Virginia 22217
and Defense Advanced Research Projects Agency
Arlington, Virginia 22209

CONTRACT N00014-76-C-0885

TM-6263/000/00

System
Development
Corporation

RESEARCH ON DEDUCTIVE INFERENCE FOR LARGE DATA BASES

FINAL TECHNICAL REPORT

Covering the Period

1 April 1976 through 30 December 1978

CHARLES KELLOGG AND IRIS KAMENY

31 JANUARY 1979

Prepared for:

Office of Naval Research, Arlington, Virginia 22217
and Defense Advanced Research Projects Agency
Arlington, Virginia 22209

CONTRACT N00014-76-C-0885

TM-6263/000/00

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE — READ INSTRUCTIONS BEFORE COMPLETING FORM

14

1. REPORT NUMBER: SDC - TM-6263/000/00
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): 6 Research on Deductive Inference for Large Data Bases.
5. TYPE OF REPORT & PERIOD COVERED: 9 Final Technical Report, 1 Apr[redacted] 76 — 30 Dec[redacted] 78
6. PERFORMING ORG. REPORT NUMBER: SDC Project RC700

7. AUTHOR(s): 10 Charles/Kellogg [redacted] Iris/Kameny
8. CONTRACT OR GRANT NUMBER(s): 15 N00014-76-C-0885, ARPA Order [redacted]-3162

9. PERFORMING ORGANIZATION NAME AND ADDRESS: System Development Corporation, 2500 Colorado Avenue, Santa Monica, CA. 90406
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Arlington, Virginia 22217
12. REPORT DATE: 11 Jan[redacted]979
13. NUMBER OF PAGES: 134

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 12 137p.
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Distribution as instructed by the Office of Naval Research

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Deductive Inference, Data Base Access, End User Facilities, Decision Aids, Natural Language Processing, Command and Control, Artificial Intelligence, DADM, EUFID.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The research summarized in this report has as its major goal the construction of software tools to aid on-line decision makers and data base users in accessing information relevant to their needs, in understanding the full data base search implications of their requests, and in reviewing and evaluating the utility of derived answers.

The conceptual framework within which this research has been carried out is based upon mathematical logic. It is becoming increasingly clear that logic is highly relevant not only to reasoning about data but to query language design,

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

339 900

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

to data structuring, to the support of high level user views, to maintaining the integrity of data bases, and to making the transition from present day data-based systems to future knowledge-based systems.

The main software tool that has been implemented as part of this research is called DADM (for Deductively Augmented Data Management). This report describes the design, implementation, and current capability of this prototype system. DADM adds a general knowledge base and a deductive processor to a data management system. These components are used to control the creation of intelligent data base access strategies and the construction of evidence to support derived answers.

The DADM prototype has been designed with logical completeness, performance and usability in mind. Completeness assures that all answers will be found. Performance has been stressed by developing new techniques for relevant premise selection, creation and verification of inference plans before data base searching, and by the use of efficient structure sharing techniques. Usability features include the use of simple structured forms for knowledge and query input, computer guidance and help when desired, and the incorporation of easy to read displays of plans, answers, and evidence. The prototype is currently operational on a DEC-10 computer in INTERLISP and on an AMDAHL 470/V6 computer in SDC LISP 1.5.

Additional work under this contract on certain aspects of the EUFID language processing system is discussed in a second section of this final report.

FOREWARD

This final report consists of two main parts. The first part describing research on deductively augmented data management (DADM) was written by Charles Kellogg. The second part describing several tasks carried out in support of the EUFID system was written by Iris Kameny.

PART I

TABLE OF CONTENTS

Figure	Title	Page

1. INTRODUCTION

The research summarized in this report has as its major goal the construction of a software tool to aid on-line decision makers and data base users in accessing information relevant to their needs, in understanding the full data base search implications of their requests, and in reviewing and evaluating the utility of derived answers.

The conceptual framework within which this research has been carried out is based upon mathematical logic. It is becoming increasingly clear that logic is highly revelant not only to reasoning about data but to query language design, to data structuring, to the support of high level user views, to maintaining the integrity of data bases, and to making the transition from present day data based systems to future knowledge based systems*.

The software tool that has been implemented as part of this research is called DADM (for Deductively Augmented Data Management) and the main body of this report describes the design, implementation, and current capability of this prototype system. The DADM environment is illustrated in figure 1 where the DADM Controller, Deductive Processor, and Answer and Evidence Generator are interfaced between a user and a relational data management system.

DADM adds a general knowledge base to a data management system. According to logic relational concepts may be specified in intensional or extensional form. Relations specified in extension correspond to the tuples or specific facts that comprise the records in a relational data base. Intensionally specified relations, on the other hand, are represented by general declarative statements (premises) and/or by computable procedures.

A major use of DADM is to quickly find intensionally specified general knowledge relevant to a user's information request and to then reason with and combine

*A recent book LOGIC and DATA BASES, Plenun Press, New York, 1978, H. Gallairé and J. Minker (Eds) presents the first comprehensive description of how logic can be used as both a practical tool and as a unifying formalism for data base system design.

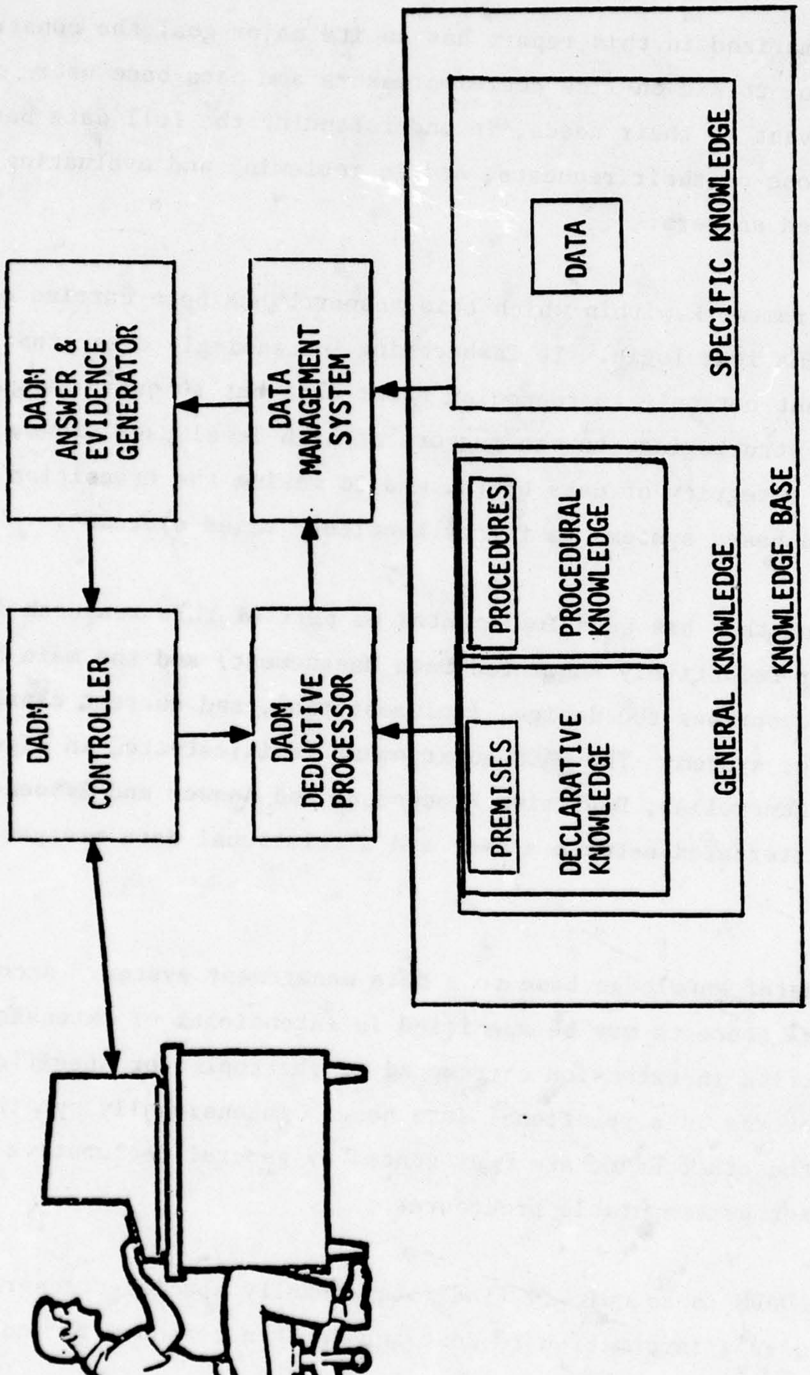


Figure 1. Deductively Augmented Data Management

this information in order to create intelligent data base access strategies and find evidence for derived answers.

As data bases become larger and more complex and serve a more diverse set of users, the ability to reason with general knowledge about the data base domain may become critical. This is because reasoning ability becomes more and more essential in bridging the gap between the high level concepts in which the user frames his query and the low level concepts which are used in describing the data base.

In implementing the DADM prototype we have emphasized logical completeness, performance, and usability. Completeness assures us that all answers will be found. Performance has been stressed by developing new techniques for relevant premise selection, creation and verification of inference plans before data base searching, and by the use of efficient structure sharing techniques. Usability features include the use of simple structured forms for knowledge and query input, computer guidance and help when desired, and the incorporation of easy to read displays of plans, answers, and evidence. The prototype is currently operational on a DEC-10 computer in INTERLISP and on an AMDAHL 470/V6 computer in SDC LISP 1.5.

2. AN ON-LINE SESSION WITH DADM

2.1 ANSWERING QUESTIONS WITH GENERAL DECLARATIVE KNOWLEDGE

DADM usually reasons with general declarative knowledge (premises) in order to create inference plans and intelligent data base access strategies (search/compute plans). In some cases, however, as shown in Figure 2, DADM can respond with specific answers derived directly from general declarative knowledge. First two elementary premises are added to the system by use of the INSERT mode. The first premise states that for every man:x and woman:y if x is the husband of y then x is married to y (Every man who is a husband is married to a woman). The second states that for every man:x, woman:y and place:p if x is married to a y who lives in p, then x also lives in p (read IF __ THEN __ for the IMP sign used in specifying premises and GIVEN __ FIND __ for its use in queries).

With this knowledge DADM can directly answer the question: Given that Socrates is husband of Xanthippe and Xanthippe lives in Athens, where does Socrates live (lives-in Socrates P)? This particular bit of reasoning is of course obvious from "common-sense". However, DADM could just as well have been searching for and combining knowledge from a much larger set of more complex premises. Since DADM is a logically complete deductive system users can be assured of getting all possible answers within the user controllable effort limit (currently set at 6 deductive paths -- see third line of printout in figure 2).

The second question in figure 2 illustrates DADM's ability to find deductive paths linking two partially specified relational concepts (HUSBAND and LIVES-IN). Variables are automatically supplied for the missing arguments and a partial (i.e., incomplete) inference plan is produced that indicates MAN-1 lives-in PLACE-1 <u>if</u> it is the case that WOMAN-1 lives-in that place (given they are married to each other and MAN-1 is the husband). DADM's ability to interpret incompletely specified queries and reason with incompletely specified knowledge (in the premise base, in the procedure base, and in the data base) is one of its unique and major strengths).

```
DADM]
ENTERED IN D.A.D.M.          VERSION 19.
CURRENT EFFORT LIMIT SETTING:  6
MODE:  ?
one of:
Query:
Insert
Delete
Adjust
Show
Find info
Garbage collect
Lisp
Exit
Keep
Teach
Help
ASsistant.
INSERT:  Premise.  Enter premise:
((MAN X)(WOMAN Y)(HUSBAND X Y)IMP(MARRIED X Y))
PREMISE INPUT FOR PREMISE  (1)  ACKNOWLEDGED.
INSERT:  Premise.  Enter premise:
((MAN X)(WOMAN Y)(PLACE P)(AND(MARRIED X Y)
(LIVES-IN Y P))
IMP
(LIVES-IN X P))
PREMISE INPUT FOR PREMISE  (2)  ACKNOWLEDGED.
INSERT:  end insert.

MODE:  Query:
.((AND(HUSBAND(SOCRATES)(XANTHIPPE))(LIVES-IN (XANTHIPPE)(ATHENS)))
IMP(LIVES-IN (SOCRATES) P))

***************
ANSWER SUMMARY --
VARIABLES:
(P)
ANSWERS:
(ATHENS)
***************

MODE:  Query:
.((HUSBAND) IMP (LIVES-IN))
(HUSBAND  HAS MISSING ARGUMENTS.    2  HAVE BEEN SUPPLIED.)
(LIVES-IN  HAS MISSING ARGUMENTS.   2  HAVE BEEN SUPPLIED.)
DEADEND SUBPROBLEMS THAT REQUIRE NEW PREMISE/TUPLE/PROCEDURE:
(LIVES-IN.2.2)
PARTIAL PLANS?Yes

1 PATHS   2 PROBLEMS   1 PLANS

NEXT?Full plans

<<INFERENCE PLAN 1 PLAUSIBILITY:  99

====================

 **1
 ASSUME HUSBAND (MAN-1 WOMAN-1)
 CONCLUDE MARRIED (MAN-1 WOMAN-1)

**0
SUPP-REQ LIVES-IN (WOMAN-1 PLACE-1)
CONCLUDE LIVES-IN (MAN-1 PLACE-1)

====================
```

Figure 2. Reasoning with premises

Before proceeding to several less elementary examples, a few words on DADM/User dialog are in order. DADM queries and premises are currently input in a formal yet simply structured language consisting of relation names, function names, domain (i.e., variable-type) names, constants (i.e., objects, numbers), variables, logical connectives (such as IMP, AND, OR, NOT) and (optionally) quantifiers (SOME, ALL). At any point in using DADM a user can type a "?" or "H" to obtain a list of (MODE, SUBMODE, etc.) options at that point or help in using the system. In the first example the top level (query, insert, delete, etc.) modes are shown. These simple syntax and explanatory features make DADM easy to learn and use.

2.2 ANSWERING QUESTIONS WITH GENERAL AND SPECIFIC KNOWLEDGE

Creation of a general declarative knowledge base can take place before or after creation of a data base. While the latter situation is most likely in practice in this next example we follow the former course since we wish to illustrate how a "classic" deduction can be carried out by DADM and related to the searching of a data base. The source of the following deduction is Sherlock Holmes "Adventure of the Dancing Men":

> "So, Watson,...You do not propose to invest in South African securities?"
> "How on earth do you know that?" I asked.
> "...It was not really difficult, by an inspection of the groove between your left forefinger and thumb, to feel sure that you did not propose to invest your small capital in the goldfields."
>
> "Here are the missing links of the very simple chain:
>
> 1. You had chalk between your left finger and thumb when you returned from the club last night.
> 2. You put chalk there when you play billiards to steady the cue.
> 3. You never play billiards except with Thurston.

4. You told me four weeks ago that Thurston had an option on some South African property which would expire in a month, and which he desired you to share with him.
5. Your cheque-book is locked in my drawer, and you have not asked for the key.
6. You do not propose to invest your money in this manner."

"How absurdly simple!" I cried.

"Quite so!" said he.

We can attribute Holmes successful deduction in this case (and many others) to his amazing ability to selectively retrieve facts from a large data base of specific world knowledge and his ability to construct plausible (yet relatively shallow) inferences from this information. In this example, for instance, Holmes needs to "search" for only two pieces of specific information: (1) the fact that Thurston wanted Watson to share his South African securities and (2) the fact that Watson did not have his cheque book.

In sentential logic -- a logic in which each relation is zero-place (i.e., has no arguments) the "Holmes" deduction can be formulated as shown below in terms of a "Query" expressing the desired conclusions, an "Inference Plan" composed of three premises, and two "Find" statements that must be shown to hold in the data base.

QUERY: If Holmes observed chalk in groove then
Holmes knew Watson did not buy securities.

INFERENCE PLAN:

PREMISE: If Holmes knew Watson played billiards with Thurston and Holmes knew Thurston wanted Watson to share securities and Holmes knew Watson did not have cheque book then Holmes knew Watson did not buy securities.

PREMISE: If Holmes observed chalk in groove then Holmes knew Watson played billiards

PREMISE: If Holmes knew Watson played billiards then Holmes knew Watson played billiards with Thurston

FIND: Holmes knew Thurston wanted Watson to share securities.

FIND: Holmes knew Watson did not have cheque book.

ANSWER: Yes

Bringing this Holmesian deduction one step closer to data base searching, we can define two one place BASE (Search) relations, one two place PROCEDURAL (compute) relation, and four one and zero place VIRTUAL (deduce) relations:

BASE (SEARCH) RELATIONS:

B1: HOLMES-KNEW-THURSTON-WANTED-WATSON-TO-SHARE-SECURITIES (DATE)

B2: HOLMES-KNEW-WATSON-DID-NOT-HAVE-CHEQUE-BOOK (DATE)

PROCEDURAL (COMPUTE) RELATION:

C1: DIFFERENCE-BETWEEN ($DATE_1$ $DATE_2$ TIME-INTERVAL)

VIRTUAL (DEDUCE) RELATIONS:

V1: HOLMES-OBSERVED-CHALK-IN-GROOVE (DATE)

V2: HOLMES-KNEW-WATSON-PLAYED-BILLIARDS (DATE)

V3: HOLMES-KNEW-WATSON-PLAYED-BILLIARDS-WITH-THURSTON (DATE)

V4: HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURITIES

The three premises shown below describe and interrelate these four virtual relations. Each of the premises is assigned a numeric plausibility weight of between 0 and 100 that may be used in computing the plausibility of inference plans and proofs (evidence chains).

PREMISES:

P1: ((HOLMES-OBSERVED-CHALK-IN-GROOVE T2)
IMP
(HOLMES-KNEW-WATSON-PLAYED-BILLIARDS T2)99)

```
P2:  ((HOLMES-KNEW-WATSON-PLAYED-BILLIARDS T2)
     IMP
     (HOLMES-KNEW-WATSON-PLAYED-BILLIARDS-WITH-THURSTON T2)99)

P3:  ((AND (HOLMES-KNEW-WATSON-PLAYED-BILLIARDS-WITH-THURSTON T2)
     (HOLMES-KNEW-THURSTON-WANTED-WATSON-TO-SHARE-SECURITIES T1)
     (HOLMES-KNEW-WATSON-DID-NOT-HAVE-CHEQUE-BOOK T2)
     (DIFFERENCE-BETWEEN T2 T1 (ONE-MONTH)))
     (IMP (HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURITIES)99)
```

Upon entering the three premises into DADM along with appropriate tuples in the data base and a LISP function to compute DIFFERENCE-BETWEEN the HOLMES query was typed in (notice spelling corrector at work) and inference and search/compute plans were produced as shown in figure 3.

DADM "answers" queries by treating them as problems to be solved. In this case there is one top level (Holmes-Knew-Watson-Did-Not-Buy-Securities) problem and three sub-problems (one compute, two search). One deductive path (middle-term chain) suffices to link together the three relevant premises into a single inference plan.

This inference plan states that in order to conclude that Watson didn't buy the securities (step **Ø), it is necessary to conclude that Watson played billiards with Thurston (step **1) and in order to reach that conclusion it is necessary to conclude that Watson played billiards (step **2). These conclusions are forthcoming if the search/compute plan shown below the inference plan can be satisfied.

That it is satisfied is shown in figure 4 where the answer (YES) and evidence chain supporting the answer is shown. It will readily be seen that an evidence chain has the same overall structure as an inference plan. Each evidence chain is an "instance" (or instantiation) of an inference plan where the inference plan's variables (e.g., THING-1) are replaced by found or computed values (e.g., JUNE 27, 1898) and SEARCH and COMPUTE are replaced by FACT and COMPUTED respectively.

```
MODE:  Query:
.((HOLMES-OBSERVED-CHALK-IN-GROOVE (JULY27,1898))
IMP
(HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURIRIES))
HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURIRIES=
HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURITIES ?  Yes

1 PATHS  4 PROBLEMS  1 PLANS

NEXT?Full plans

<<INFERENCE PLAN 1 PLAUSIBILITY:  99
====================

  **2
  ASSUME HOLMES-OBSERVED-CHALK-IN-GROOVE (JULY27,1898)
  CONCLUDE HOLMES-KNEW-WATSON-PLAYED-BILLIARDS (JULY27,1898)

 **1
 CONCLUDE HOLMES-KNEW-WATSON-PLAYED-BILLIARDS-WITH-THURSTON (JULY27,1898)

**0
COMPUTE DIFFERENCE-BETWEEN (JULY27,1898 THING-1 ONE-MONTH)
SEARCH HOLMES-KNEW-WATSON-DID-NOT-HAVE-CHEQUE-BOOK (JULY27,1898)
SEARCH HOLMES-KNEW-THURSTON-WANTED-WATSON-TO-SHARE-SECURITIES (THING-1)
CONCLUDE HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURITIES
====================

SEARCH/COMPUTE PLAN:
    SEARCH      *HOLMES-KNEW-THURSTON-WANTED-WATSON-TO-SHARE-SECURITIES
                THING-1
    SEARCH      *HOLMES-KNEW-WATSON-DID-NOT-HAVE-CHEQUE-BOOK
                JULY27,1898
    COMPUTE     *DIFFERENCE-BETWEEN JULY27,1898 THING-1 ONE-MONTH
```

Figure 3. Relating a classic deduction to data base searching

```
ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
YES
****************

EVIDENCE CHAIN 1 FROM PLAN 1 PLAUSIBILITY:  99

====================

  **2
  ASSUME HOLMES-OBSERVED-CHALK-IN-GROOVE (JULY27,1898)
  CONCLUDE HOLMES-KNEW-WATSON-PLAYED-BILLIARDS (JULY27,1898)

 **1
 CONCLUDE HOLMES-KNEW-WATSON-PLAYED-BILLIARDS-WITH-THURSTON (JULY27,1898)


**0
COMPUTED DIFFERENCE-BETWEEN (JULY27,1898 JUNE27,1898 ONE-MONTH)
FACT HOLMES-KNEW-WATSON-DID-NOT-HAVE-CHEQUE-BOOK (JULY27,1898)
FACT HOLMES-KNEW-THURSTON-WANTED-WATSON-TO-SHARE-SECURITIES (JUNE27,1898)

CONCLUDE HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURITIES
====================
                                                    >>

NEXT?Usage flow.  Enter plan number or list of plan numbers:
1


PLAN 1
STEP WT   USES
**2  99   PREMISE 3
**1  99   **2 PREMISE 5
**0  99   **1 PREMISE 4
NEXT?
Done.
**
```

Figure 4. Relating a classic deduction to data base search (continued)

```
MODE:  ASsistant.
_USE ()
...FOR (HOLMES-OBS$ --)
...
=HOLMES-OBSERVED-CHALK-IN-GROOVE
DEADEND SUBPROBLEMS THAT REQUIRE NEW PREMISE/TUPLE/PROCEDURE:
(HOLMES-OBSERVED-CHALK-IN-GROOVE.3.1)
PARTIAL PLANS?Yes

3 PATHS  7 PROBLEMS  1 PLANS

NEXT?Full plans

<<INFERENCE PLAN 1 PLAUSIBILITY:  99
====================

  **2
  SUPP-REQ HOLMES-OBSERVED-CHALK-IN-GROOVE (THING-2)
  CONCLUDE HOLMES-KNEW-WATSON-PLAYED-BILLIARDS (THING-2)

 **1
 CONCLUDE HOLMES-KNEW-WATSON-PLAYED-BILLIARDS-WITH-THURSTON (THING-2)

**0
COMPUTE DIFFERENCE-BETWEEN (THING-2 THING-1 ONE-MONTH)
SEARCH HOLMES-KNEW-WATSON-DID-NOT-HAVE-CHEQUE-BOOK (THING-2)
SEARCH HOLMES-KNEW-THURSTON-WANTED-WATSON-TO-SHARE-SECURITIES (THING-1)
CONCLUDE HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURITIES

====================


SEARCH/COMPUTE PLAN:
    SEARCH        *HOLMES-KNEW-THURSTON-WANTED-WATSON-TO-SHARE-SECURITIES
                  THING-1
    SEARCH        *HOLMES-KNEW-WATSON-DID-NOT-HAVE-CHEQUE-BOOK THING-2
    COMPUTE       *DIFFERENCE-BETWEEN THING-2 THING-1 ONE-MONTH
```

DADM keeps a history list of all user inputs that is accessible by the assistant. In this example the assistant is instructed to replace the given clause (HOLMES-OBS---) by () turning the GIVEN -- FIND -- query into a FIND -- type query. We end up with a deadend subproblem, a partial plan (note SUPPORT-REQUIRED in step **2) and a SEARCH/COMPUTE plan in two variables (THING-1, THING-2).

Figure 5. Finding deductive support for a FIND type question.

```
MODE:  ASsistant.
 USE ()
... FOR (HOLMES-KN$ --)
...IN (HOLMES-OB$ --)
...
=HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURITIES

3 PATHS  7 PROBLEMS  1 PLANS

NEXT?Full plans

<<INFERENCE PLAN 1 PLAUSIBILITY:  99

====================

MAIN FORWARD CHAINS:
--------------------
**1
ASSUME HOLMES-OBSERVED-CHALK-IN-GROOVE (THING-2)
CONCLUDE HOLMES-KNEW-WATSON-PLAYED-BILLIARDS (THING-2)

 **2
 CONCLUDE HOLMES-KNEW-WATSON-PLAYED-BILLIARDS-WITH-THURSTON (THING-2)

  **3
  SEARCH HOLMES-KNEW-THURSTON-WANTED-WATSON-TO-SHARE-SECURITIES (THING-1
**)

  SEARCH HOLMES-KNEW-WATSON-DID-NOT-HAVE-CHEQUE-BOOK (THING-2)
  COMPUTE DIFFERENCE-BETWEEN (THING-2 THING-1 ONE-MONTH)
  CONCLUDE HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURITIES

====================
```

The assistant is used again to convert the GIVEN -- FIND -- type query into a GIVEN -- type query. In this situation DADM reasons from the given assumption forward through the premises until a complete plan is constructed or until a deductive limit is reached. GIVEN type queries as especially useful for testing working hypotheses and for generalized navigation (browsing) through virtual relations.

Figure 6. Finding deductive consequences for a GIVEN type question.

After the NEXT? prompt the user types "U" (the rest of the characters being supplied by DADM type ahead) and then "1". DADM responds with a list of steps in the plan, the plausibility weight associated with each step, and the premises and previous steps used in deriving each step. Usage flow information has been separated from plan/evidence information because many times it is not of interest to users.

2.3 DERIVE ALTERNATIVE COURSES OF ACTION TO SUPPORT ON-LINE DECISION MAKING: REASONING ABOUT COMMAND AND CONTROL

DADM's ability to reason about data can be a considerable aid to the on-line decision maker. Given a question (problem) to be answered (solved) DADM can quickly select relevant modular chunks of declarative knowledge and combine them into plans which upon execution produce answers and evidence articulating the alternative courses of action (ACOA's) open to the decision maker. These ACOA's will, of course, only be as good or relevant as the knowledge in the knowledge base. Since DADM adds new dimensions of descriptive, deductive, and planning capabilities to on-line data base searching it has the potential for becoming a major new form of on-line decision aid. As a short example of the possibilities in this area, consider a Task Force Commander who asks the question: "How can I achieve ASW-SCREEN ready status if the Peterson returns to port?" The Commander's aide would normally have to formulate a series of complex requests about the Peterson's function in the screen, the availability of suitable replacement ships, their ready status etc. Using DADM the question could be formulated as shown in figure 7. DADM then displays the full inferential and search implications of the request which include the Peterson leaving the Task-force, causing a gap in the ASW-SCREEN and a CONFIGURATION-1 type hole that must be filled by a SHIP-1 that has the right equipment, status, etc.

The complexity of the DADM produced search plan is further illustrated in figure 8. Here the Intermediate Language (IL) control mode is turned on, the question repeated (REDO Q), and a lengthy relational algebra form of

```
MODE:  Query:
.((RETURNS (PETERSEN) (PORT)) IMP(STATUS (ASW-SCREEN) (READY)))

2 PATHS  11 PROBLEMS  1 PLANS

NEXT?Full plans

<<INFERENCE PLAN 1 PLAUSIBILITY:  99

====================

  **3
  SEARCH READY-STATUS (SHIP-1 READY)
  SEARCH AVAILABLE-FOR-ASSIGNMENT (SHIP-1 CONFIGURATION-1)
  SEARCH ASW-EQUIPMENT (CONFIGURATION-1 ASROC)
  SEARCH CLASS (CONFIGURATION-1 ADAMS)
  SEARCH TYPE (CONFIGURATION-1 DD)
  CONCLUDE AVAILABLE-FOR-ASW (SHIP-1 CONFIGURATION-1)

     **6
     ASSUME RETURNS (PETERSEN PORT)
     CONCLUDE LEAVES (PETERSEN TASK-FORCE)

    **5
    CONCLUDE NOT PART-OF (PETERSEN TASK-FORCE)

   **4
   SEARCH EMPLOYED-IN (PETERSEN ASW-SCREEN)
   CONCLUDE CAUSE-GAP (PETERSEN ASW-SCREEN)

  **2
  SEARCH ASSIGNED-TO (PETERSEN CONFIGURATION-1)
  CONCLUDE HOLE (CONFIGURATION-1 ASW-SCREEN)

 **1
 SEARCH AUTHORIZE-TRANSFER (CAPTAIN SHIP-1 ASW-SCREEN)
 CONCLUDE FILL-HOLE (SHIP-1 ASW-SCREEN)

**0
SEARCH COMPLETE (ASW-SCREEN)
CONCLUDE STATUS (ASW-SCREEN READY)

====================


SEARCH/COMPUTE PLAN:
    SEARCH      *EMPLOYED-IN PETERSEN ASW-SCREEN
    SEARCH      *ASSIGNED-TO PETERSEN CONFIGURATION-1
    SEARCH      *TYPE CONFIGURATION-1 DD
    SEARCH      *CLASS CONFIGURATION-1 ADAMS
    SEARCH      *ASW-EQUIPMENT CONFIGURATION-1 ASROC
    SEARCH      *AVAILABLE-FOR-ASSIGNMENT SHIP-1 CONFIGURATION-1
    SEARCH      *READY-STATUS SHIP-1 READY
    SEARCH      *AUTHORIZE-TRANSFER CAPTAIN SHIP-1 ASW-SCREEN
    SEARCH      *COMPLETE ASW-SCREEN
```

Figure 7. Alternative courses of action - Implications of Request.

```
ADJUST:  Control.  Enter list of control modes:

(IL)
OK
ADJUST:  end adjustment.

**
MODE:  ASsistant.
_REDO Q

2 PATHS  11 PROBLEMS  1 PLANS

NEXT?PLan display.  Enter plan number or list of plan numbers:
1
PLAN NUMBER: 1

SEARCH/COMPUTE PLAN:
    SEARCH      *EMPLOYED-IN PETERSEN ASW-SCREEN
    SEARCH      *ASSIGNED-TO PETERSEN CONFIGURATION-1
    SEARCH      *TYPE CONFIGURATION-1 DD
    SEARCH      *CLASS CONFIGURATION-1 ADAMS
    SEARCH      *ASW-EQUIPMENT CONFIGURATION-1 ASROC
    SEARCH      *AVAILABLE-FOR-ASSIGNMENT SHIP-1 CONFIGURATION-1
    SEARCH      *READY-STATUS SHIP-1 READY
    SEARCH      *AUTHORIZE-TRANSFER CAPTAIN SHIP-1 ASW-SCREEN
    SEARCH      *COMPLETE ASW-SCREEN
EXECUTE?Yes
retrieve[ASSIGNED-TO.CONFIGURATION,AVAILABLE-FOR-ASSIGNMENT.SHIP]
  where(EMPLOYED-IN.SHIP="PETERSEN")
  and(EMPLOYED-IN.FUNCTION="ASW-SCREEN")
  and(ASSIGNED-TO.SHIP="PETERSEN")
  and(TYPE.CONFIGURATION=ASSIGNED-TO.CONFIGURATION)
  and(TYPE.CATEGORY="DD")
  and(CLASS.CONFIGURATION=ASSIGNED-TO.CONFIGURATION)
  and(CLASS.SHIP="ADAMS")
  and(ASW-EQUIPMENT.CONFIGURATION=ASSIGNED-TO.CONFIGURATION)
  and(ASW-EQUIPMENT.MISSILE="ASROC")
  and(AVAILABLE-FOR-ASSIGNMENT.CONFIGURATION=ASSIGNED-TO.CONFIGURATION)
  and(READY-STATUS.SHIP=AVAILABLE-FOR-ASSIGNMENT.SHIP)
  and(READY-STATUS.VALUE="READY")
  and(AUTHORIZE-TRANSFER.RANK="CAPTAIN")
  and(AUTHORIZE-TRANSFER.SHIP=AVAILABLE-FOR-ASSIGNMENT.SHIP)
  and(AUTHORIZE-TRANSFER.FUNCTION="ASW-SCREEN")
  and(COMPLETE.FUNCTION="ASW-SCREEN")
```

Figure 8. Alternative courses of action - Search strategy for external data base.

```
ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL
****************
ANSWER SUMMARY --
CONDITIONAL ANSWERS:
YES IF --
               *AUTHORIZE-TRANSFER CAPTAIN SPRUANCE ASW-SCREEN
               *COMPLETE ASW-SCREEN
YES IF --
               *AUTHORIZE-TRANSFER CAPTAIN KINKAID ASW-SCREEN
               *COMPLETE ASW-SCREEN
****************

EVIDENCE CHAIN 1 FROM PLAN 1 PLAUSIBILITY:  99

====================

  **3
  FACT READY-STATUS (SPRUANCE READY)
  FACT AVAILABLE-FOR-ASSIGNMENT (SPRUANCE ATTACK-MODE)
  FACT ASW-EQUIPMENT (ATTACK-MODE ASROC)
  FACT CLASS (ATTACK-MODE ADAMS)
  FACT TYPE (ATTACK-MODE DD)
  CONCLUDE AVAILABLE-FOR-ASW (SPRUANCE ATTACK-MODE)

     **6
     ASSUME RETURNS (PETERSEN PORT)
     CONCLUDE LEAVES (PETERSEN TASK-FORCE)

    **5
    CONCLUDE NOT PART-OF (PETERSEN TASK-FORCE)

   **4
   FACT EMPLOYED-IN (PETERSEN ASW-SCREEN)
   CONCLUDE CAUSE-GAP (PETERSEN ASW-SCREEN)

  **2
  FACT ASSIGNED-TO (PETERSEN ATTACK-MODE)
  CONCLUDE HOLE (ATTACK-MODE ASW-SCREEN)

 **1
 FACT-REQ AUTHORIZE-TRANSFER (CAPTAIN SPRUANCE ASW-SCREEN)
 CONCLUDE FILL-HOLE (SPRUANCE ASW-SCREEN)

**0
FACT-REQ COMPLETE (ASW-SCREEN)
CONCLUDE STATUS (ASW-SCREEN READY)

====================

NEXT?Usage flow.  Enter plan number or list of plan numbers:
1

PLAN 1
STEP WT   USES
**3  99   PREMISE 12
**6  99   PREMISE 10
**5  99   **6 PREMISE 7
**4  99   **5 PREMISE 9
**2  99   **4 PREMISE 8
**1  99   **2 **3 PREMISE 6
**0  99   **1 PREMISE 11
NEXT?
Done.
```

Figure 9. Alternative courses of action - Conditional answers and evidence.

```
**
MODE:  Query:
.(()IMP(STATUS))
(STATUS  HAS MISSING ARGUMENTS.   2  HAVE BEEN SUPPLIED.)

CHAINS LIMIT REACHED
FURTHER DEDUCTION REQUIRED:
(LEAVES.7.1)
TRY HARDER?Yes
TRYING HARDER:
DEADEND SUBPROBLEMS THAT REQUIRE NEW PREMISE/TUPLE/PROCEDURE:
(RETURNS.10.1)
PARTIAL PLANS?Yes

7 PATHS  17 PROBLEMS  1 PLANS

NEXT?Full plans

<<INFERENCE PLAN 1 PLAUSIBILITY:  99

====================

  **3
  SEARCH READY-STATUS (SHIP-2 READY)
  SEARCH AVAILABLE-FOR-ASSIGNMENT (SHIP-2 CONFIGURATION-1)
  SEARCH ASW-EQUIPMENT (CONFIGURATION-1 ASROC)
  SEARCH CLASS (CONFIGURATION-1 ADAMS)
  SEARCH TYPE (CONFIGURATION-1 DD)
  CONCLUDE AVAILABLE-FOR-ASW (SHIP-2 CONFIGURATION-1)

      **6
      SUPP-REQ RETURNS (SHIP-1 PORT)
      CONCLUDE LEAVES (SHIP-1 TASK-FORCE)

     **5
     CONCLUDE NOT PART-OF (SHIP-1 TASK-FORCE)

    **4
    SEARCH EMPLOYED-IN (SHIP-1 ASW-SCREEN)
    CONCLUDE CAUSE-GAP (SHIP-1 ASW-SCREEN)

   **2
   SEARCH ASSIGNED-TO (SHIP-1 CONFIGURATION-1)
   CONCLUDE HOLE (CONFIGURATION-1 ASW-SCREEN)

  **1
  SEARCH AUTHORIZE-TRANSFER (CAPTAIN SHIP-2 ASW-SCREEN)
  CONCLUDE FILL-HOLE (SHIP-2 ASW-SCREEN)

**0
SEARCH COMPLETE (ASW-SCREEN)
CONCLUDE STATUS (ASW-SCREEN READY)

====================


SEARCH/COMPUTE PLAN:
    SEARCH      *EMPLOYED-IN SHIP-1 ASW-SCREEN
    SEARCH      *ASSIGNED-TO SHIP-1 CONFIGURATION-1
    SEARCH      *TYPE CONFIGURATION-1 DD
    SEARCH      *CLASS CONFIGURATION-1 ADAMS
    SEARCH      *ASW-EQUIPMENT CONFIGURATION-1 ASROC
    SEARCH      *AVAILABLE-FOR-ASSIGNMENT SHIP-2 CONFIGURATION-1
    SEARCH      *READY-STATUS SHIP-2 READY
    SEARCH      *AUTHORIZE-TRANSFER CAPTAIN SHIP-2 ASW-SCREEN
    SEARCH      *COMPLETE ASW-SCREEN
```

Figure 10. Alternative courses of action - Incompletely specified FIND type question.

```
DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
CONDITIONAL ANSWERS:
YES IF --
                *AUTHORIZE-TRANSFER CAPTAIN BRISCOE ASW-SCREEN
                *COMPLETE ASW-SCREEN
YES IF --
                *AUTHORIZE-TRANSFER CAPTAIN SPRUANCE ASW-SCREEN
                *COMPLETE ASW-SCREEN
YES IF --
                *AUTHORIZE-TRANSFER CAPTAIN KINKAID ASW-SCREEN
                *COMPLETE ASW-SCREEN
****************

EVIDENCE CHAIN 1 FROM PLAN 1 PLAUSIBILITY:  99

===================

  **3
  FACT READY-STATUS (BRISCOE READY)
  FACT AVAILABLE-FOR-ASSIGNMENT (BRISCOE DEFENSE-MODE)
  FACT ASW-EQUIPMENT (DEFENSE-MODE ASROC)
  FACT CLASS (DEFENSE-MODE ADAMS)
  FACT TYPE (DEFENSE-MODE DD)
  CONCLUDE AVAILABLE-FOR-ASW (BRISCOE DEFENSE-MODE)

     **6
     SUPP-REQ RETURNS (JOHN-HANCOCK PORT)
     CONCLUDE LEAVES (JOHN-HANCOCK TASK-FORCE)

    **5
    CONCLUDE NOT PART-OF (JOHN-HANCOCK TASK-FORCE)

   **4
   FACT EMPLOYED-IN (JOHN-HANCOCK ASW-SCREEN)
   CONCLUDE CAUSE-GAP (JOHN-HANCOCK ASW-SCREEN)

  **2
  FACT ASSIGNED-TO (JOHN-HANCOCK DEFENSE-MODE)
  CONCLUDE HOLE (DEFENSE-MODE ASW-SCREEN)

 **1
 FACT-REQ AUTHORIZE-TRANSFER (CAPTAIN BRISCOE ASW-SCREEN)
 CONCLUDE FILL-HOLE (BRISCOE ASW-SCREEN)

 **0
 FACT-REQ COMPLETE (ASW-SCREEN)
 CONCLUDE STATUS (ASW-SCREEN READY)

===================
```

Figure 11. Alternative courses of action - The Briscoe can replace the John-Hancock if the latter returns to port.

search strategy is generated. IL search requests of this form will eventually be sent to external data management systems over a network connection.

Figure 9 shows the several alternatives that have been found as a result of deductively guided data base search. Two ships, the Spruance and the Kinkaid, have been located that conditionally satisfy the intent of the original request. The condition is that the Captain (Commander) must authorize transfer of the ship to the screen and there must be no other holes in the screen (it must be complete). Conditional answers illustrate yet another important aspect of DADM's ability to return useful results in the face of incomplete information. The conditional evidence chain for the first answer is shown where FACT-REQ indicates facts that are required to convert the chain from conditional to complete status.

The usage flow at the bottom of figure 9 illustrates how the steps in a moderately complex deduction are derived from various premises and preceeding steps.

Figures 10 to 13 are included to show backward reasoning (figures 10, 11), forward reasoning (figure 12), and reasoning with negation (figure 13) variations on the original command and control query. Note the utility of the TRY HARDER and PARTIAL PLAN facilities for coping with incomplete plans.

2.4 DERIVING MULTIPLE CHAINS OF EVIDENCE TO SUPPORT HIGH LEVEL CONJECTURES: REASONING ABOUT SCIENTIFIC COMMUNICATION

In the last section we demonstrated how DADM produced answers and evidence chains could be interpreted as distinct patterns of information representing alternative courses of action. In this section we will illustrate how DADM produced inference plans and search strategies of varying plausibility may produce multiple threads of evidence relevant to the same high level conjecture. In many cases these threads of evidence have a mutually reinforcing effect

that can markedly improve on-line performance in judging how strongly the evidence supports or refutes the user's conjecture. We will also see that DADM generates inference paths, answers, and evidence in a most plausible, shortest path order so the most concise and credible information is viewed first.

Consider a bibliographic data base that contains in addition to the usual author, publication, citing, and subject matter relationships other information on scientist-authors such as the name and location of their research laboratories, information about their academic backgrounds, and information about their attendance at various scientific conferences. The two queries in figure 14 are typical requests of such a data base. The first request provides a list of scientists and their laboratories by year and country, while the second reveals that Barker, who studied under Wilkins, is the author of a series of publications on bubble memory technology.

Now let us suppose that an analyst familiar with this data base wants to find out if a certain scientific result achieved in 1978 but not yet published in the general scientific literature may also be known by research workers at British laboratories. Notice our use of "may" in the last sentence. It is unlikely that our analyst can establish directly, given the kind of data base he has, that a British laboratory knew about the particular result. However, through the use of mechanized inference, he may be able to build a rather strong body of evidence to support a conjecture to that effect.

In order to respond to queries of this form, premises must be formulated that somehow relate information about the originator of a result to scientists and laboratories that may know about the result. Premises and relations relevant to this problem have been defined and entered into DADM as shown in the printout of DADM's inventory of relations, domains, and premise names shown in figure 15. First we see the HUSBAND, MARRIED,

```
Query:
.(()IMP(AND(CONDUCTS-RESEARCH-AT SCI LAB YR)(LOCATED-IN LAB CTRY)))

0 PATHS  2 PROBLEMS  1 PLANS

NEXT?Full plans

SEARCH/COMPUTE PLAN:
    SEARCH       *CONDUCTS-RESEARCH-AT THING-SCI THING-LAB THING-YR
    SEARCH       *LOCATED-IN THING-LAB THING-CTRY

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

***************
ANSWER SUMMARY --
VARIABLES:
(SCI LAB YR CTRY)
ANSWERS:
(AUDLEY-CHARLES STRATHCLYDE 1978 UK)
(BARTON-BROWNE MIT 1950 USA)
(BARTON-BROWNE LANCASTER 1978 UK)
(MACKENZIE CAMBRIDGE 1978 UK)
(SMITH IMPERIAL-COLLEGE 1978 UK)
(KILLICK-KENDRICK LANCASTER 1978 UK)
(HALLIDAY EDINBURGH 1978 UK)
(SOUTHWOOD CAMBRIDGE 1978 UK)
***************

MODE:  Query:
.(()IMP(AND(STUDIED-UNDER X (WILKINS))(AUTHOR X PUBS)))

0 PATHS  2 PROBLEMS  1 PLANS

NEXT?Full plans

SEARCH/COMPUTE PLAN:
    SEARCH       *STUDIED-UNDER THING-X WILKINS
    SEARCH       *AUTHOR THING-X THING-PUBS

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

***************
ANSWER SUMMARY --
VARIABLES:
(X PUBS)
ANSWERS:
(BARKER VISCOUS-FLOW-IN-BUBBLE-MEMORIES)
(BARKER FUNDAMENTALS-OF-BUBBLE-MEMORIES)
(BARKER LATTICE-ARCHITECTURE-FOR-BUBBLE-WALL-STORAGE)
(BARKER FABRICATION-OF-BUBBLE-CHIPS-USING-G3)
***************
```

Figure 14. Multiple chains of evidence: Base Relation Searching.

```
MODE:  Show
SHOW:  Relation tree.
(RELATIONS (HUSBAND)
           (MARRIED)
           (LIVES-IN)
           (HOLMES-OBSERVED-CHALK-IN-GROOVE)
           (HOLMES-KNEW-WATSON-PLAYED-BILLIARDS)
           (HOLMES-KNEW-WATSON-PLAYED-BILLIARDS-WITH-THURSTON)
           (HOLMES-KNEW-THURSTON-WANTED-WATSON-TO-SHARE-SECURITIES)
           (HOLMES-KNEW-WATSON-DID-NOT-HAVE-CHEQUE-BOOK)
           (DIFFERENCE-BETWEEN)
           (HOLMES-KNEW-WATSON-DID-NOT-BUY-SECURITIES)
           (IC-VIRTUAL-RELATIONS (ORIGINATES)
                                 (MEMBER-SAME-IC)
                                 (SCIENTIFIC-INFORMATION-FLOW)
                                 (KNOWS))
           (IC-BASE-RELATIONS (STUDIED-UNDER)
                              (MASTER-TEACHER)
                              (ABOUT)
                              (ATTEND)
                              (CONFERENCE-ON)
                              (CONDUCTS-RESEARCH-AT)
                              (LOCATED-IN)
                              (AUTHOR)
                              (CITES))
           (ASW-VIRTUAL-RELATIONS (RETURNS)
                                  (CAUSE-GAP)
                                  (PART-OF)
                                  (LEAVES)
                                  (FILL-HOLE)
                                  (HOLE)
                                  (AVAILABLE-FOR-ASW)
                                  (STATUS))
           (ASW-BASE-RELATIONS (COMPLETE)
                               (AUTHORIZE-TRANSFER)
                               (READY-STATUS)
                               (AVAILABLE-FOR-ASSIGNMENT)
                               (EMPLOYED-IN)
                               (ASSIGNED-TO)
                               (TYPE)
                               (CLASS)
                               (ASW-EQUIPMENT))

SHOW:  Domain tree.            SHOW:  Premise tree.
(DOMAINS (MAN)                 (PREMISES (STUDENT-MBR)
         (WOMAN)                         (AUTHOR-MBR)
         (PLACE)                         (CITE-AUTHOR-MBR)
         (CONFIGURATION)                 (CITE-PUB-MBR)
         (SHIP)                          (ORIG-IC)
         (LOC)                           (CONF-IC)
         (SCIENTIST)                     (IC-LAB)
         (PUBLICATION)                   (IC-IC))
         (RESULT)              SHOW:  end show.
         (YEAR)
         (MEETING)
         (SUBJECT)
         (LAB))
```

Figure 15. Multiple chains of evidence: Relation, Domain, and Premise Names.

LIVES-IN, and HOLMES relations along with the ASW example BASE and VIRTUAL relations. Next is a list of new "IC" BASE and VIRTUAL relations where IC stands for the notion of Invisible College (i.e., scientific in-group or clique). To support the IC VIRTUAL relations (ORIGINATES, MEMBER-SAME-IC, SCIENTIFIC-INFORMATION-FLOW, KNOWS) we have constructed eight premises, and given them the names shown near the bottom of figure 15. Four of these premises formalize criteria for membership in an invisible college and the rest relate scientists and laboratories to knowledge shared by the members of an invisible college. Two sample premises are expressed in English below:

> SCIENTISTS WHO CO-AUTHOR A PUBLICATION MAY BE MEMBERS OF THE SAME INVISIBLE COLLEGE.

> A SCIENTIST WHO ORIGINATES A NEW RESULT DURING A YEAR IS LIKELY TO TRANSMIT KNOWLEDGE OF THAT RESULT TO MEMBERS OF HIS INVISIBLE COLLEGE DURING THAT YEAR.

In figure 16 the conjecture that UK laboratories know about a magnetic bubble result originated by Barker is input to DADM and 6 deductive paths and plans requiring solution of 21 problems and subproblems are found before the chains limit is reached. The initial usage flow for the 6 plans is displayed and we immediately notice that plans 1 and 4 have the highest plausibility weight.

In figures 17 to 24 the plans and evidence chains are shown in user preferred order and format. In the next figure the final usage flow information details the plausibility weights and deductive support for the steps in each of the 6 plans.

In figures 26 through 30 the DADM system "trys harder" and two additional but less plausible plans are found. Then the IL control mode is turned on to demonstrate the generation of more complex forms of IL search requests. Next the four invisible college membership premises are deleted in order to

```
MODE:  Query:
.((LAB L)
(ORIGINATES (BARKER) (MAG-BUBBLE) (1978))
IMP
(AND (KNOWS L (MAG-BUBBLE) (1978)) (LOCATED-IN L (UK))))

CHAINS LIMIT REACHED

6 PATHS  21 PROBLEMS  6 PLANS

NEXT?Usage flow.  Enter plan number or list of plan numbers:
(1 TO 6)

PLAN 1
WT   PREMISES
95   (16 17 19)

PLAN 2
WT   PREMISES
80   (15 17 19)

PLAN 3
WT   PREMISES
70   (14 17 19)

PLAN 4
WT   PREMISES
95   (16 17 18 19 20)

PLAN 5
WT   PREMISES
80   (15 17 18 19 20)

PLAN 6
WT   PREMISES
70   (14 17 18 19 20)
```

Figure 16. Multiple chains of evidence: The conjecture and initial usage flow.

```
NEXT?PLan display.  Enter plan number or list of plan numbers:
(1 4)
PLAN NUMBER: 1

<<INFERENCE PLAN 1 PLAUSIBILITY:  95

2 SUBPLANS:
====================

  **2
  SEARCH CITES (PUBLICATION-1 PUBLICATION-2)
  SEARCH CITES (PUBLICATION-2 PUBLICATION-1)
  SEARCH AUTHOR (SCIENTIST-1 PUBLICATION-1)
  SEARCH AUTHOR (BARKER PUBLICATION-2)
  CONCLUDE MEMBER-SAME-IC (BARKER SCIENTIST-1)

 **1
 ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
 CONCLUDE KNOWS (SCIENTIST-1 MAG-BUBBLE 1978)

**0
SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-1 LAB-L 1978)
CONCLUDE KNOWS (LAB-L MAG-BUBBLE 1978)

====================

SEARCH LOCATED-IN (LAB-L UK)

====================

SEARCH/COMPUTE PLAN:
    SEARCH      *AUTHOR BARKER PUBLICATION-2
    SEARCH      *AUTHOR SCIENTIST-1 PUBLICATION-1
    SEARCH      *CITES PUBLICATION-2 PUBLICATION-1
    SEARCH      *CITES PUBLICATION-1 PUBLICATION-2
    SEARCH      *CONDUCTS-RESEARCH-AT SCIENTIST-1 LAB-L 1978
    SEARCH      *LOCATED-IN LAB-L UK
```

Figure 17. Multiple chains of evidence: Plan-1.

```
EXECUTE?Yes

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

***************
ANSWER SUMMARY --
VARIABLES:
(L)
ANSWERS:
(EDINBURGH)
***************

EVIDENCE CHAIN 1 FROM PLAN 1 PLAUSIBILITY:  95

2 CONCLUSIONS:
===================

  **2
  FACT CITES (BUBBLE-MEMORIES-REVISITED VISCOUS-FLOW-IN-BUBBLE-MEMORIES)
  FACT CITES (VISCOUS-FLOW-IN-BUBBLE-MEMORIES BUBBLE-MEMORIES-REVISITED)
  FACT AUTHOR (HALLIDAY BUBBLE-MEMORIES-REVISITED)
  FACT AUTHOR (BARKER VISCOUS-FLOW-IN-BUBBLE-MEMORIES)
  CONCLUDE MEMBER-SAME-IC (BARKER HALLIDAY)

 **1
 ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
 CONCLUDE KNOWS (HALLIDAY MAG-BUBBLE 1978)

**0
FACT CONDUCTS-RESEARCH-AT (HALLIDAY EDINBURGH 1978)
CONCLUDE KNOWS (EDINBURGH MAG-BUBBLE 1978)

===================


FACT LOCATED-IN (EDINBURGH UK)

===================
                                        >>
```

Figure 18. Multiple chains of evidence: Chain-1.

```
PLAN NUMBER: 4

<<INFERENCE PLAN 4 PLAUSIBILITY:  95

2 SUBPLANS:
====================

  **3
  SEARCH CONFERENCE-ON (MEETING-1 SUBJECT-1 1978)
  SEARCH ATTEND (SCIENTIST-1 MEETING-1 1978)
  SEARCH ATTEND (SCIENTIST-2 MEETING-1 1978)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (SCIENTIST-2 SCIENTIST-1 SUBJECT-1 1978)

   **4
   SEARCH CITES (PUBLICATION-1 PUBLICATION-2)
   SEARCH CITES (PUBLICATION-2 PUBLICATION-1)
   SEARCH AUTHOR (SCIENTIST-2 PUBLICATION-1)
   SEARCH AUTHOR (BARKER PUBLICATION-2)
   CONCLUDE MEMBER-SAME-IC (BARKER SCIENTIST-2)

  **2
  ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
  CONCLUDE KNOWS (SCIENTIST-2 MAG-BUBBLE 1978)

 **1
 SEARCH ABOUT (MAG-BUBBLE SUBJECT-1)
 CONCLUDE KNOWS (SCIENTIST-1 MAG-BUBBLE 1978)

**0
SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-1 LAB-L 1978)
CONCLUDE KNOWS (LAB-L MAG-BUBBLE 1978)

====================

SEARCH LOCATED-IN (LAB-L UK)

====================


SEARCH/COMPUTE PLAN:
    SEARCH       *AUTHOR BARKER PUBLICATION-2
    SEARCH       *AUTHOR SCIENTIST-2 PUBLICATION-1
    SEARCH       *CITES PUBLICATION-2 PUBLICATION-1
    SEARCH       *CITES PUBLICATION-1 PUBLICATION-2
    SEARCH       *ABOUT MAG-BUBBLE SUBJECT-1
    SEARCH       *ATTEND SCIENTIST-2 MEETING-1 1978
    SEARCH       *ATTEND SCIENTIST-1 MEETING-1 1978
    SEARCH       *CONFERENCE-ON MEETING-1 SUBJECT-1 1978
    SEARCH       *CONDUCTS-RESEARCH-AT SCIENTIST-1 LAB-L 1978
    SEARCH       *LOCATED-IN LAB-L UK
EXECUTE?Yes
```

Figure 19. Multiple chains of evidence: Plan-4

```
DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
VARIABLES:
(L)
ANSWERS:
(CAMBRIDGE)
****************

EVIDENCE CHAIN 1 FROM PLAN 4 PLAUSIBILITY:  95

2 CONCLUSIONS:
===================

   **3
   FACT CONFERENCE-ON (APS BUBBLE-MEMORIES 1978)
   FACT ATTEND (SOUTHWOOD APS 1978)
   FACT ATTEND (BOYCE APS 1978)
   CONCLUDE SCIENTIFIC-INFORMATION-FLOW (BOYCE SOUTHWOOD BUBBLE-MEMORIES 1978)

    **4
    FACT CITES (HIGH-SPEED-BUBBLE-MEMORIES VISCOUS-FLOW-IN-BUBBLE-MEMORIES)
    FACT CITES (VISCOUS-FLOW-IN-BUBBLE-MEMORIES HIGH-SPEED-BUBBLE-MEMORIES)
    FACT AUTHOR (BOYCE HIGH-SPEED-BUBBLE-MEMORIES)
    FACT AUTHOR (BARKER VISCOUS-FLOW-IN-BUBBLE-MEMORIES)
    CONCLUDE MEMBER-SAME-IC (BARKER BOYCE)

   **2
   ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
   CONCLUDE KNOWS (BOYCE MAG-BUBBLE 1978)

  **1
  FACT ABOUT (MAG-BUBBLE BUBBLE-MEMORIES)
  CONCLUDE KNOWS (SOUTHWOOD MAG-BUBBLE 1978)

 **0
 FACT CONDUCTS-RESEARCH-AT (SOUTHWOOD CAMBRIDGE 1978)
 CONCLUDE KNOWS (CAMBRIDGE MAG-BUBBLE 1978)

 ===================

 FACT LOCATED-IN (CAMBRIDGE UK)

 ===================
```

Figure 20. Multiple chains of evidence: Chain-4.

```
NEXT?Execute plan.  Enter plan number or list of plan numbers:
(2 3 5 6)
PLAN NUMBER: 2
SEARCH/COMPUTE PLAN:
     SEARCH      *AUTHOR BARKER PUBLICATION-2
     SEARCH      *AUTHOR SCIENTIST-1 PUBLICATION-1
     SEARCH      *CITES PUBLICATION-2 SCIENTIST-1
     SEARCH      *CITES PUBLICATION-1 BARKER
     SEARCH      *CONDUCTS-RESEARCH-AT SCIENTIST-1 LAB-L 1978
     SEARCH      *LOCATED-IN LAB-L UK

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL
****************
ANSWER SUMMARY --
VARIABLES:
(L)
ANSWERS:
(STRATHCLYDE)
****************

EVIDENCE CHAIN 1 FROM PLAN 2 PLAUSIBILITY:  8Ø

2 CONCLUSIONS:
====================

  **2
  FACT CITES (CAGE-MATERIAL-FOR-BUBBLE-MEMORIES BARKER)
  FACT CITES (FABRICATION-OF-BUBBLE-CHIPS-USING-G3 AUDLEY-CHARLES)
  FACT AUTHOR (AUDLEY-CHARLES CAGE-MATERIAL-FOR-BUBBLE-MEMORIES)
  FACT AUTHOR (BARKER FABRICATION-OF-BUBBLE-CHIPS-USING-G3)
  CONCLUDE MEMBER-SAME-IC (BARKER AUDLEY-CHARLES)

 **1
 ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
 CONCLUDE KNOWS (AUDLEY-CHARLES MAG-BUBBLE 1978)

**Ø
FACT CONDUCTS-RESEARCH-AT (AUDLEY-CHARLES STRATHCLYDE 1978)
CONCLUDE KNOWS (STRATHCLYDE MAG-BUBBLE 1978)

====================

FACT LOCATED-IN (STRATHCLYDE UK)

====================
```

Figure 21. Multiple chains of evidence: Search plan and chain-2.

```
PLAN NUMBER: 3
SEARCH/COMPUTE PLAN:
    SEARCH      *AUTHOR BARKER PUBLICATION-1
    SEARCH      *AUTHOR SCIENTIST-1 PUBLICATION-1
    SEARCH      *CONDUCTS-RESEARCH-AT SCIENTIST-1 LAB-L 1978
    SEARCH      *LOCATED-IN LAB-L UK

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

***************
ANSWER SUMMARY --
VARIABLES:
(L)
ANSWERS:
(LANCASTER)
***************

EVIDENCE CHAIN 1 FROM PLAN 3 PLAUSIBILITY:  70

2 CONCLUSIONS:
===================

  **2
  FACT AUTHOR (KILLICK-KENDRICK FUNDAMENTALS-OF-BUBBLE-MEMORIES)
  FACT AUTHOR (BARKER FUNDAMENTALS-OF-BUBBLE-MEMORIES)
  CONCLUDE MEMBER-SAME-IC (BARKER KILLICK-KENDRICK)

 **1
 ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
 CONCLUDE KNOWS (KILLICK-KENDRICK MAG-BUBBLE 1978)

**0
FACT CONDUCTS-RESEARCH-AT (KILLICK-KENDRICK LANCASTER 1978)
CONCLUDE KNOWS (LANCASTER MAG-BUBBLE 1978)

===================

FACT LOCATED-IN (LANCASTER UK)

===================

                                        >>
```

Figure 22. Multiple chains of evidence: Search plan and chain-3.

```
PLAN NUMBER: 5
SEARCH/COMPUTE PLAN:
    SEARCH       *AUTHOR BARKER PUBLICATION-2
    SEARCH       *AUTHOR SCIENTIST-2 PUBLICATION-1
    SEARCH       *CITES PUBLICATION-2 SCIENTIST-2
    SEARCH       *CITES PUBLICATION-1 BARKER
    SEARCH       *ABOUT MAG-BUBBLE SUBJECT-1
    SEARCH       *ATTEND SCIENTIST-2 MEETING-1 1978
    SEARCH       *ATTEND SCIENTIST-1 MEETING-1 1978
    SEARCH       *CONFERENCE-ON MEETING-1 SUBJECT-1 1978
    SEARCH       *CONDUCTS-RESEARCH-AT SCIENTIST-1 LAB-L 1978
    SEARCH       *LOCATED-IN LAB-L UK
ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

***************
ANSWER SUMMARY --
VARIABLES:
(L)
ANSWERS:
(LANCASTER)
(CAMBRIDGE)
***************

EVIDENCE CHAIN 1 FROM PLAN 5 PLAUSIBILITY:  80

2 CONCLUSIONS:
===================

  **3
  FACT CONFERENCE-ON (IEEE BUBBLE-MEMORIES 1978)
  FACT ATTEND (BARTON-BROWNE IEEE 1978)
  FACT ATTEND (YOSHIDA IEEE 1978)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (YOSHIDA BARTON-BROWNE BUBBLE-MEMORIES 1978)

   **4
   FACT CITES (CONTIGUOUS-DISK-BUBBLE-MEMORIES BARKER)
   FACT CITES (LATTICE-ARCHITECTURE-FOR-BUBBLE-WALL-STORAGE YOSHIDA)
   FACT AUTHOR (YOSHIDA CONTIGUOUS-DISK-BUBBLE-MEMORIES)
   FACT AUTHOR (BARKER LATTICE-ARCHITECTURE-FOR-BUBBLE-WALL-STORAGE)
   CONCLUDE MEMBER-SAME-IC (BARKER YOSHIDA)

  **2
  ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
  CONCLUDE KNOWS (YOSHIDA MAG-BUBBLE 1978)

 **1
 FACT ABOUT (MAG-BUBBLE BUBBLE-MEMORIES)
 CONCLUDE KNOWS (BARTON-BROWNE MAG-BUBBLE 1978)

**0
FACT CONDUCTS-RESEARCH-AT (BARTON-BROWNE LANCASTER 1978)
CONCLUDE KNOWS (LANCASTER MAG-BUBBLE 1978)

===================


FACT LOCATED-IN (LANCASTER UK)

===================
```

Figure 23. Multiple chains of evidence: Search plan and chain-5.

```
PLAN NUMBER: 6
SEARCH/COMPUTE PLAN:
     SEARCH      *AUTHOR BARKER PUBLICATION-1
     SEARCH      *AUTHOR SCIENTIST-2 PUBLICATION-1
     SEARCH      *ABOUT MAG-BUBBLE SUBJECT-1
     SEARCH      *ATTEND SCIENTIST-2 MEETING-1 1978
     SEARCH      *ATTEND SCIENTIST-1 MEETING-1 1978
     SEARCH      *CONFERENCE-ON MEETING-1 SUBJECT-1 1978
     SEARCH      *CONDUCTS-RESEARCH-AT SCIENTIST-1 LAB-L 1978
     SEARCH      *LOCATED-IN LAB-L UK

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
VARIABLES:
(L)
ANSWERS:
(CAMBRIDGE)
****************

EVIDENCE CHAIN 1 FROM PLAN 6 PLAUSIBILITY:  70

2 CONCLUSIONS:
====================

   **3
   FACT CONFERENCE-ON (APS BUBBLE-MEMORIES 1978)
   FACT ATTEND (SOUTHWOOD APS 1978)
   FACT ATTEND (BOYCE APS 1978)
   CONCLUDE SCIENTIFIC-INFORMATION-FLOW (BOYCE SOUTHWOOD BUBBLE-MEMORIES 1978)

    **4
    FACT AUTHOR (BOYCE VISCOUS-FLOW-IN-BUBBLE-MEMORIES)
    FACT AUTHOR (BARKER VISCOUS-FLOW-IN-BUBBLE-MEMORIES)
    CONCLUDE MEMBER-SAME-IC (BARKER BOYCE)

   **2
   ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
   CONCLUDE KNOWS (BOYCE MAG-BUBBLE 1978)

  **1
  FACT ABOUT (MAG-BUBBLE BUBBLE-MEMORIES)
  CONCLUDE KNOWS (SOUTHWOOD MAG-BUBBLE 1978)

**0
FACT CONDUCTS-RESEARCH-AT (SOUTHWOOD CAMBRIDGE 1978)
CONCLUDE KNOWS (CAMBRIDGE MAG-BUBBLE 1978)

====================

FACT LOCATED-IN (CAMBRIDGE UK)

====================
```

Figure 24. Multiple chains of evidence: Search plan and chain-6.

```
NEXT?Usage flow.  Enter plan number or list of plan numbers:
(1 TO 6)

PLAN 1
STEP WT    USES
**2  95    PREMISE 16
**1  99    **2 PREMISE 17
**0  99    **1 PREMISE 19

PLAN 2
STEP WT    USES
**2  80    PREMISE 15
**1  99    **2 PREMISE 17
**0  99    **1 PREMISE 19

PLAN 3
STEP WT    USES
**2  70    PREMISE 14
**1  99    **2 PREMISE 17
**0  99    **1 PREMISE 19

PLAN 4
STEP WT    USES
**3  99    PREMISE 18
**4  95    PREMISE 16
**2  99    **4 PREMISE 17
**1  99    **2 **3 PREMISE 20
**0  99    **1 PREMISE 19

PLAN 5
STEP WT    USES
**3  99    PREMISE 18
**4  80    PREMISE 15
**2  99    **4 PREMISE 17
**1  99    **2 **3 PREMISE 20
**0  99    **1 PREMISE 19

PLAN 6
STEP WT    USES
**3  99    PREMISE 18
**4  70    PREMISE 14
**2  99    **4 PREMISE 17
**1  99    **2 **3 PREMISE 20
**0  99    **1 PREMISE 19
NEXT?Try harder
TRYING HARDER:
```

Figure 25. Multiple chains of evidence: Final usage flow for first 6 plans.

```
7 PATHS  24 PROBLEMS  8 PLANS

NEXT?PLan display.  Enter plan number or list of plan numbers:
(7 8)
PLAN NUMBER: 7

<<INFERENCE PLAN 7 PLAUSIBILITY:  70

2 SUBPLANS:
====================

  **3
  SEARCH CONFERENCE-ON (MEETING-1 SUBJECT-1 1978)
  SEARCH ATTEND (SCIENTIST-1 MEETING-1 1978)
  SEARCH ATTEND (SCIENTIST-2 MEETING-1 1978)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (SCIENTIST-2 SCIENTIST-1 SUBJECT-1 1978)

   **4
   SEARCH AUTHOR (SCIENTIST-2 PUBLICATION-1)
   SEARCH AUTHOR (BARKER PUBLICATION-1)
   CONCLUDE MEMBER-SAME-IC (BARKER SCIENTIST-2)

  **2
  ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
  CONCLUDE KNOWS (SCIENTIST-2 MAG-BUBBLE 1978)

 **1
 SEARCH ABOUT (MAG-BUBBLE SUBJECT-1)
 CONCLUDE KNOWS (SCIENTIST-1 MAG-BUBBLE 1978)

**0
SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-1 LAB-L 1978)
CONCLUDE KNOWS (LAB-L MAG-BUBBLE 1978)
====================


SEARCH LOCATED-IN (LAB-L UK)

====================


SEARCH/COMPUTE PLAN:
    SEARCH      *AUTHOR BARKER PUBLICATION-1
    SEARCH      *AUTHOR SCIENTIST-2 PUBLICATION-1
    SEARCH      *ABOUT MAG-BUBBLE SUBJECT-1
    SEARCH      *ATTEND SCIENTIST-2 MEETING-1 1978
    SEARCH      *ATTEND SCIENTIST-1 MEETING-1 1978
    SEARCH      *CONFERENCE-ON MEETING-1 SUBJECT-1 1978
    SEARCH      *CONDUCTS-RESEARCH-AT SCIENTIST-1 LAB-L 1978
    SEARCH      *LOCATED-IN LAB-L UK
```

Figure 26. Multiple chains of evidence: Plan-7.

```
EXECUTE?Yes
put[retrieve[AUTHOR.SCIENTIST,AUTHOR.TITLE]]into __0082;
put[retrieve[ATTEND.SCIENTIST,ATTEND.CONFERENCE,ATTEND.DATE]]into __0083;
retrieve[AUTHOR.TITLE, __0082.0,ABOUT.MAJOR-AREA,ATTEND.CONFERENCE,__0083.0,
CONDUCTS-RESEARCH-AT.LOCATION]
  where(AUTHOR.SCIENTIST="BARKER")
  and(__0082.1=AUTHOR.TITLE)
  and(ABOUT.TOPIC="MAG-BUBBLE")
  and(ATTEND.SCIENTIST=__0082.0)
  and(ATTEND.DATE=1978)
  and(__0083.1=ATTEND.CONFERENCE)
  and(__0083.2=1978)
  and(CONFERENCE-ON.CONFERENCE=ATTEND.CONFERENCE)
  and(CONFERENCE-ON.TOPIC=ABOUT.MAJOR-AREA)
  and(CONFERENCE-ON.YEAR=1978)
  and(CONDUCTS-RESEARCH-AT.SCIENTIST=__0083.0)
  and(CONDUCTS-RESEARCH-AT.YEAR=1978)
  and(LOCATED-IN.PLACE1=CONDUCTS-RESEARCH-AT.LOCATION)
  and(LOCATED-IN.PLACE2="UK")

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
VARIABLES:
(L)
ANSWERS:
(CAMBRIDGE)
****************


EVIDENCE CHAIN 1 FROM PLAN 7 PLAUSIBILITY:  70

2 CONCLUSIONS:
====================

  **3
  FACT CONFERENCE-ON (APS BUBBLE-MEMORIES 1978)
  FACT ATTEND (SOUTHWOOD APS 1978)
  FACT ATTEND (BOYCE APS 1978)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (BOYCE SOUTHWOOD BUBBLE-MEMORIES 1978)

   **4
   FACT AUTHOR (BOYCE VISCOUS-FLOW-IN-BUBBLE-MEMORIES)
   FACT AUTHOR (BARKER VISCOUS-FLOW-IN-BUBBLE-MEMORIES)
   CONCLUDE MEMBER-SAME-IC (BARKER BOYCE)

  **2
  ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
  CONCLUDE KNOWS (BOYCE MAG-BUBBLE 1978)

 **1
 FACT ABOUT (MAG-BUBBLE BUBBLE-MEMORIES)
 CONCLUDE KNOWS (SOUTHWOOD MAG-BUBBLE 1978)

**0
FACT CONDUCTS-RESEARCH-AT (SOUTHWOOD CAMBRIDGE 1978)
CONCLUDE KNOWS (CAMBRIDGE MAG-BUBBLE 1978)

====================


FACT LOCATED-IN (CAMBRIDGE UK)

====================
```

Figure 27. Multiple chains of evidence: IL Request and chain-7.

```
PLAN NUMBER: 8

<<INFERENCE PLAN 8 PLAUSIBILITY:  60

2 SUBPLANS:
====================

  **3
  SEARCH CONFERENCE-ON (MEETING-1 SUBJECT-1 1978)
  SEARCH ATTEND (SCIENTIST-1 MEETING-1 1978)
  SEARCH ATTEND (SCIENTIST-3 MEETING-1 1978)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (SCIENTIST-3 SCIENTIST-1 SUBJECT-1 1978)

   **4
   SEARCH MASTER-TEACHER (SCIENTIST-2)
   SEARCH STUDIED-UNDER (SCIENTIST-3 SCIENTIST-2)
   SEARCH STUDIED-UNDER (BARKER SCIENTIST-2)
   CONCLUDE MEMBER-SAME-IC (BARKER SCIENTIST-3)

  **2
  ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
  CONCLUDE KNOWS (SCIENTIST-3 MAG-BUBBLE 1978)

 **1
 SEARCH ABOUT (MAG-BUBBLE SUBJECT-1)
 CONCLUDE KNOWS (SCIENTIST-1 MAG-BUBBLE 1978)

**0
SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-1 LAB-L 1978)
CONCLUDE KNOWS (LAB-L MAG-BUBBLE 1978)

====================

SEARCH LOCATED-IN (LAB-L UK)

====================
```

Figure 28. Multiple chains of evidence: Plan-8.

```
SEARCH/COMPUTE PLAN:
     SEARCH       *STUDIED-UNDER BARKER SCIENTIST-2
     SEARCH       *STUDIED-UNDER SCIENTIST-3 SCIENTIST-2
     SEARCH       *MASTER-TEACHER SCIENTIST-2
     SEARCH       *ABOUT MAG-BUBBLE SUBJECT-1
     SEARCH       *ATTEND SCIENTIST-3 MEETING-1 1978
     SEARCH       *ATTEND SCIENTIST-1 MEETING-1 1978
     SEARCH       *CONFERENCE-ON MEETING-1 SUBJECT-1 1978
     SEARCH       *CONDUCTS-RESEARCH-AT SCIENTIST-1 LAB-L 1978
     SEARCH       *LOCATED-IN LAB-L UK
EXECUTE?Yes
MASTER-TEACHER IS NOT AN EXTERNAL RELATION.
put[retrieve[STUDIED-UNDER.STUDENT,STUDIED-UNDER.TEACHER]]into __0084;
put[retrieve[ATTEND.SCIENTIST,ATTEND.CONFERENCE,ATTEND.DATE]]into __0085;
retrieve[STUDIED-UNDER.TEACHER,__0084.0,ABOUT.MAJOR-AREA,ATTEND.CONFERENCE,__0085.0,
CONDUCTS-RESEARCH-AT.LOCATION]
  where(STUDIED-UNDER.STUDENT="BARKER")
  and(__0084.1=STUDIED-UNDER.TEACHER)
  and(MASTER-TEACHER.1=STUDIED-UNDER.TEACHER)
  and(ABOUT.TOPIC="MAG-BUBBLE")
  and(ATTEND.SCIENTIST=__0084.0)
  and(ATTEND.DATE=1978)
  and(__0085.1=ATTEND.CONFERENCE)
  and(__0085.2=1978)
  and(CONFERENCE-ON.CONFERENCE=ATTEND.CONFERENCE)
  and(CONFERENCE-ON.TOPIC=ABOUT.MAJOR-AREA)
  and(CONFERENCE-ON.YEAR=1978)
  and(CONDUCTS-RESEARCH-AT.SCIENTIST=__0085.0)
  and(CONDUCTS-RESEARCH-AT.YEAR=1978)
  and(LOCATED-IN.PLACE1=CONDUCTS-RESEARCH-AT.LOCATION)
  and(LOCATED-IN.PLACE2="UK")

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL
```

Figure 29. Multiple chains of evidence: IL Request for plan-8.

```
***************
ANSWER SUMMARY --
VARIABLES:
(L)
CONDITIONAL ANSWERS:
(LANCASTER)
   INFORMATION NEEDED:
                *MASTER-TEACHER WILKINS
(CAMBRIDGE)
   INFORMATION NEEDED:
                *MASTER-TEACHER WILKINS
***************

EVIDENCE CHAIN 1 FROM PLAN 8 PLAUSIBILITY:  60
2 CONCLUSIONS:
====================
  **3
  FACT CONFERENCE-ON (IEEE BUBBLE-MEMORIES 1978)
  FACT ATTEND (BARTON-BROWNE IEEE 1978)
  FACT ATTEND (HOFFMAN IEEE 1978)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (HOFFMAN BARTON-BROWNE BUBBLE-MEMORIES 1978)

   **4
   FACT-REQ MASTER-TEACHER (WILKINS)
   FACT STUDIED-UNDER (HOFFMAN WILKINS)
   FACT STUDIED-UNDER (BARKER WILKINS)
   CONCLUDE MEMBER-SAME-IC (BARKER HOFFMAN)

  **2
  ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
  CONCLUDE KNOWS (HOFFMAN MAG-BUBBLE 1978)

 **1
 FACT ABOUT (MAG-BUBBLE BUBBLE-MEMORIES)
 CONCLUDE KNOWS (BARTON-BROWNE MAG-BUBBLE 1978)

**0
FACT CONDUCTS-RESEARCH-AT (BARTON-BROWNE LANCASTER 1978)
CONCLUDE KNOWS (LANCASTER MAG-BUBBLE 1978)

====================

FACT LOCATED-IN (LANCASTER UK)

====================

                                        >>
```

Figure 30. Multiple chains of evidence: Chain-8.

illustrate the results of searching for evidence with partial plans. (figures 31 to 33).

The use of DADM as a powerful tool for "generalized navigation" among relational concepts is explored in figures 34 through 40.

A review of these examples will, we believe, provide the reader with an appreciation of the utility of adding virtual relations to a data base and describing them in a declarative form as premises that can be combined according to the rules of symbolic logic.

2.5 REASONING WITH COMPUTABLE FUNCTIONS

The descriptive and deductive capabilities of DADM are further expanded as illustrated in figure 41 by the addition of computable functions as arguments to relations. In general, DADM will replace a computable function by its value as soon as it can be evaluated i.e., as soon as its arguments are all constants. This means that some functions will be evaluated before data base search and others (such as those in figure 41) must be evaluated after suitable values are supplied by data base search.

```
MODE:  Delete
DELETE:  Premise.  Enter premise or list of premise names or numbers:
(STUDENT-MBR AUTHOR-MBR CITE-AUTHOR-MBR CITE-PUB-MBR)
PREMISE:  STUDENT-MBR:
(((ALL (SCIENTIST . X64)) (ALL (SCIENTIST . X65)) (ALL (SCIENTIST . X66))
 (AND (STUDIED-UNDER X64 X66) (STUDIED-UNDER X65 X66) (MASTER-TEACHER X66))
 IMP (MEMBER-SAME-IC X64 X65))
STUDENT-MBR)
  DELETED.
PREMISE:  AUTHOR-MBR:
(((ALL (SCIENTIST . X67)) (ALL (SCIENTIST . X68)) (ALL (PUBLICATION . X69))
 (AND (AUTHOR X67 X69) (AUTHOR X68 X69))
 IMP (MEMBER-SAME-IC X67 X68))
AUTHOR-MBR)
  DELETED.
PREMISE:  CITE-AUTHOR-MBR:
(((ALL (SCIENTIST . X70)) (ALL (SCIENTIST . X71)) (ALL (PUBLICATION . X72))
 (ALL (PUBLICATION . X73))
 (AND (AUTHOR X70 X72) (AUTHOR X71 X73) (CITES X72 X71) (CITES X73 X70))
 IMP (MEMBER-SAME-IC X70 X71))
CITE-AUTHOR-MBR)
  DELETED.
PREMISE:  CITE-PUB-MBR:
(((ALL (SCIENTIST . X74)) (ALL (SCIENTIST . X75)) (ALL (PUBLICATION . X76))
 (ALL (PUBLICATION . X77))
 (AND (AUTHOR X74 X76) (AUTHOR X75 X77) (CITES X76 X77) (CITES X77 X76))
 IMP (MEMBER-SAME-IC X74 X75))
CITE-PUB-MBR)
  DELETED.
RELATION:  MASTER-TEACHER  DELETED.
DELETE:  end delete.
```

Figure 31. Multiple chains of evidence: Deletion of 4 premises.

```
MODE:  ASsistant.
_ REDO Q


DEADEND SUBPROBLEMS THAT REQUIRE NEW PREMISE/TUPLE/PROCEDURE:
(MEMBER-SAME-IC.17.2)
PARTIAL PLANS?Yes

3 PATHS  11 PROBLEMS  2 PLANS

NEXT?Full plans

ADJUST:  Control.  Enter list of control modes:

()
OK
ADJUST:  end adjustment.
<<INFERENCE PLAN 1 PLAUSIBILITY: 99

2 SUBPLANS:
====================

**1
 SUPP-REQ MEMBER-SAME-IC (BARKER SCIENTIST-1)
 ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
 CONCLUDE KNOWS (SCIENTIST-1 MAG-BUBBLE 1978)

**0
SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-1 LAB-L 1978)
CONCLUDE KNOWS (LAB-L MAG-BUBBLE 1978)

====================


SEARCH LOCATED-IN (LAB-L UK)

====================


SEARCH/COMPUTE PLAN:
    SEARCH      *CONDUCTS-RESEARCH-AT SCIENTIST-1 LAB-L 1978
    SEARCH      *LOCATED-IN LAB-L UK

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
VARIABLES:
(L)
ANSWERS:
(STRATHCLYDE)
(IMPERIAL-COLLEGE)
(LANCASTER)
(EDINBURGH)
(CAMBRIDGE)
****************
```

Figure 32. Multiple chains of evidence: REDO of Conjecture; Partial Plan-1.

```
<<INFERENCE PLAN 2 PLAUSIBILITY:  99

2 SUBPLANS:
====================

  **3
  SEARCH CONFERENCE-ON (MEETING-1 SUBJECT-1 1978)
  SEARCH ATTEND (SCIENTIST-1 MEETING-1 1978)
  SEARCH ATTEND (SCIENTIST-2 MEETING-1 1978)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (SCIENTIST-2 SCIENTIST-1 SUBJECT-1 1978)

  **2
  SUPP-REQ MEMBER-SAME-IC (BARKER SCIENTIST-2)
  ASSUME ORIGINATES (BARKER MAG-BUBBLE 1978)
  CONCLUDE KNOWS (SCIENTIST-2 MAG-BUBBLE 1978)

 **1
 SEARCH ABOUT (MAG-BUBBLE SUBJECT-1)
 CONCLUDE KNOWS (SCIENTIST-1 MAG-BUBBLE 1978)

**0
SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-1 LAB-L 1978)
CONCLUDE KNOWS (LAB-L MAG-BUBBLE 1978)

====================


SEARCH LOCATED-IN (LAB-L UK)

====================


SEARCH/COMPUTE PLAN:
    SEARCH      *ABOUT MAG-BUBBLE SUBJECT-1
    SEARCH      *ATTEND SCIENTIST-2 MEETING-1 1978
    SEARCH      *ATTEND SCIENTIST-1 MEETING-1 1978
    SEARCH      *CONFERENCE-ON MEETING-1 SUBJECT-1 1978
```

Figure 33. Multiple chains of evidence: Partial Plan-2.

```
MODE:
Query:
.((ORIGINATES)IMP(KNOWS))
(ORIGINATES  HAS MISSING ARGUMENTS.    3  HAVE BEEN SUPPLIED.)
(KNOWS  HAS MISSING ARGUMENTS.    3  HAVE BEEN SUPPLIED.)
DEADEND SUBPROBLEMS THAT REQUIRE NEW PREMISE/TUPLE/PROCEDURE:
(MEMBER-SAME-IC.17.2)
PARTIAL PLANS?Yes

5 PATHS  14 PROBLEMS  4 PLANS

NEXT?PLan display.  Enter plan number or list of plan numbers:
 (1 2 3 4)
 PLAN NUMBER: 1

 <<INFERENCE PLAN 1 PLAUSIBILITY:  99

 ====================

 **0
 SUPP-REQ MEMBER-SAME-IC (SCIENTIST-2 SCIENTIST-1)
 ASSUME ORIGINATES (SCIENTIST-2 RESULT-1 YEAR-1)
 CONCLUDE KNOWS (SCIENTIST-1 RESULT-1 YEAR-1)

 ====================


 EXECUTE?No
```

Figure 34. Multiple chains of evidence: Generalized Navigation; ORIGINATES --- KNOWS, Plan-1

```
PLAN NUMBER: 2

<<INFERENCE PLAN 2 PLAUSIBILITY:  99

===================

 **1
 SUPP-REQ MEMBER-SAME-IC (SCIENTIST-1 SCIENTIST-2)
 ASSUME ORIGINATES (SCIENTIST-1 RESULT-1 YEAR-1)
 CONCLUDE KNOWS (SCIENTIST-2 RESULT-1 YEAR-1)

**0
SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-2 LAB-1 YEAR-1)
CONCLUDE KNOWS (LAB-1 RESULT-1 YEAR-1)

===================


SEARCH/COMPUTE PLAN:
     SEARCH      *CONDUCTS-RESEARCH-AT SCIENTIST-2 LAB-1 YEAR-1
EXECUTE?Yes

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
YES
****************


EVIDENCE CHAIN 1 FROM PLAN 2 PLAUSIBILITY:  99

===================

 **1
 SUPP-REQ MEMBER-SAME-IC (SCIENTIST-1 SOUTHWOOD)
 ASSUME ORIGINATES (SCIENTIST-1 RESULT-1 1978)
 CONCLUDE KNOWS (SOUTHWOOD RESULT-1 1978)
**0
FACT CONDUCTS-RESEARCH-AT (SOUTHWOOD CAMBRIDGE 1978)
CONCLUDE KNOWS (CAMBRIDGE RESULT-1 1978)

===================
                                                      >>
```

Figure 35. Multiple chains of evidence: Plan-2.

```
PLAN NUMBER: 3

<<INFERENCE PLAN 3 PLAUSIBILITY:  99

====================

 **2
 SEARCH CONFERENCE-ON (MEETING-1 SUBJECT-1 YEAR-1)
 SEARCH ATTEND (SCIENTIST-3 MEETING-1 YEAR-1)
 SEARCH ATTEND (SCIENTIST-1 MEETING-1 YEAR-1)
 CONCLUDE SCIENTIFIC-INFORMATION-FLOW (SCIENTIST-1 SCIENTIST-3 SUBJECT-1 YEAR-1)

 **1
 SUPP-REQ MEMBER-SAME-IC (SCIENTIST-2 SCIENTIST-1)
 ASSUME ORIGINATES (SCIENTIST-2 RESULT-1 YEAR-1)
 CONCLUDE KNOWS (SCIENTIST-1 RESULT-1 YEAR-1)

**0
SEARCH ABOUT (RESULT-1 SUBJECT-1)
CONCLUDE KNOWS (SCIENTIST-3 RESULT-1 YEAR-1)

====================


SEARCH/COMPUTE PLAN:
    SEARCH      *ABOUT RESULT-1 SUBJECT-1
    SEARCH      *ATTEND SCIENTIST-1 MEETING-1 YEAR-1
    SEARCH      *ATTEND SCIENTIST-3 MEETING-1 YEAR-1
    SEARCH      *CONFERENCE-ON MEETING-1 SUBJECT-1 YEAR-1
EXECUTE?Yes

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
YES
****************


EVIDENCE CHAIN 1 FROM PLAN 3 PLAUSIBILITY:  99

====================

 **2
 FACT CONFERENCE-ON (APS BUBBLE-MEMORIES 1978)
 FACT ATTEND (SOUTHWOOD APS 1978)
 FACT ATTEND (BOYCE APS 1978)
 CONCLUDE SCIENTIFIC-INFORMATION-FLOW (BOYCE SOUTHWOOD BUBBLE-MEMORIES 1978)

 **1
 SUPP-REQ MEMBER-SAME-IC (SCIENTIST-2 BOYCE)
 ASSUME ORIGINATES (SCIENTIST-2 MAG-BUBBLE 1978)
 CONCLUDE KNOWS (BOYCE MAG-BUBBLE 1978)

**0
FACT ABOUT (MAG-BUBBLE BUBBLE-MEMORIES)
CONCLUDE KNOWS (SOUTHWOOD MAG-BUBBLE 1978)

====================

                                        >>
```

Figure 36. Multiple chains of evidence: Plan-3

```
PLAN NUMBER: 4

<<INFERENCE PLAN 4 PLAUSIBILITY:  99

===================

  **3
  SEARCH CONFERENCE-ON (MEETING-1 SUBJECT-1 YEAR-1)
  SEARCH ATTEND (SCIENTIST-2 MEETING-1 YEAR-1)
  SEARCH ATTEND (SCIENTIST-3 MEETING-1 YEAR-1)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (SCIENTIST-3 SCIENTIST-2 SUBJECT-1 YEAR-1)

  **2
  SUPP-REQ MEMBER-SAME-IC (SCIENTIST-1 SCIENTIST-3)
  ASSUME ORIGINATES (SCIENTIST-1 RESULT-1 YEAR-1)
  CONCLUDE KNOWS (SCIENTIST-3 RESULT-1 YEAR-1)

 **1
 SEARCH ABOUT (RESULT-1 SUBJECT-1)
 CONCLUDE KNOWS (SCIENTIST-2 RESULT-1 YEAR-1)

**0
SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-2 LAB-1 YEAR-1)
CONCLUDE KNOWS (LAB-1 RESULT-1 YEAR-1)

===================


SEARCH/COMPUTE PLAN:
    SEARCH      *ABOUT RESULT-1 SUBJECT-1
    SEARCH      *ATTEND SCIENTIST-3 MEETING-1 YEAR-1
    SEARCH      *ATTEND SCIENTIST-2 MEETING-1 YEAR-1
    SEARCH      *CONFERENCE-ON MEETING-1 SUBJECT-1 YEAR-1
    SEARCH      *CONDUCTS-RESEARCH-AT SCIENTIST-2 LAB-1 YEAR-1
DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
YES
****************

EVIDENCE CHAIN 1 FROM PLAN 4 PLAUSIBILITY:  99

===================

  **3
  FACT CONFERENCE-ON (IEEE BUBBLE-MEMORIES 1978)
  FACT ATTEND (MACKENZIE IEEE 1978)
  FACT ATTEND (YOSHIDA IEEE 1978)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (YOSHIDA MACKENZIE BUBBLE-MEMORIES 1978)

  **2
  SUPP-REQ MEMBER-SAME-IC (SCIENTIST-1 YOSHIDA)
  ASSUME ORIGINATES (SCIENTIST-1 MAG-BUBBLE 1978)
  CONCLUDE KNOWS (YOSHIDA MAG-BUBBLE 1978)

 **1
 FACT ABOUT (MAG-BUBBLE BUBBLE-MEMORIES)
 CONCLUDE KNOWS (MACKENZIE MAG-BUBBLE 1978)

**0
FACT CONDUCTS-RESEARCH-AT (MACKENZIE CAMBRIDGE 1978)
CONCLUDE KNOWS (CAMBRIDGE MAG-BUBBLE 1978)
```

Figure 37. Multiple chains of evidence: Plan-4

```
MODE: Query:
.((KNOWS) IMP (KNOWS))
(KNOWS  HAS MISSING ARGUMENTS.    3  HAVE BEEN SUPPLIED.)
(KNOWS  HAS MISSING ARGUMENTS.    3  HAVE BEEN SUPPLIED.)

5 PATHS  11 PROBLEMS  4 PLANS

NEXT?PLan display.  Enter plan number or list of plan numbers:
(1 2   4)
PLAN NUMBER: 1

<<INFERENCE PLAN 1 PLAUSIBILITY:  100

===================

**0
ASSUME KNOWS (THING-2 THING-1 THING-3)
CONCLUDE KNOWS (THING-2 THING-1 THING-3)

===================


EXECUTE?No
PLAN NUMBER: 2

<<INFERENCE PLAN 2 PLAUSIBILITY:  99

===================

**0
SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-1 LAB-1 YEAR-1)
ASSUME KNOWS (SCIENTIST-1 RESULT-1 YEAR-1)
CONCLUDE KNOWS (LAB-1 RESULT-1 YEAR-1)

===================


SEARCH/COMPUTE PLAN:
    SEARCH       *CONDUCTS-RESEARCH-AT SCIENTIST-1 LAB-1 YEAR-1
EXECUTE?Yes

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

***************
ANSWER SUMMARY --
YES
***************

EVIDENCE CHAIN 1 FROM PLAN 2 PLAUSIBILITY:  99

===================

**0
FACT CONDUCTS-RESEARCH-AT (SOUTHWOOD CAMBRIDGE 1978)
ASSUME KNOWS (SOUTHWOOD RESULT-1 1978)
CONCLUDE KNOWS (CAMBRIDGE RESULT-1 1978)

===================
```

Figure 38. Multiple chains of evidence: Recursive navigation; Plan-1, Plan-2.

```
<<INFERENCE PLAN 4 PLAUSIBILITY:  99

====================

  **2
  SEARCH CONFERENCE-ON (MEETING-1 SUBJECT-1 YEAR-1)
  SEARCH ATTEND (SCIENTIST-2 MEETING-1 YEAR-1)
  SEARCH ATTEND (SCIENTIST-1 MEETING-1 YEAR-1)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (SCIENTIST-1 SCIENTIST-2 SUBJECT-1 YEAR-1)

 **1
 SEARCH ABOUT (RESULT-1 SUBJECT-1)
 ASSUME KNOWS (SCIENTIST-1 RESULT-1 YEAR-1)
 CONCLUDE KNOWS (SCIENTIST-2 RESULT-1 YEAR-1)

**0
SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-2 LAB-1 YEAR-1)
CONCLUDE KNOWS (LAB-1 RESULT-1 YEAR-1)

====================


SEARCH/COMPUTE PLAN:
    SEARCH      *ABOUT RESULT-1 SUBJECT-1
    SEARCH      *ATTEND SCIENTIST-1 MEETING-1 YEAR-1
    SEARCH      *ATTEND SCIENTIST-2 MEETING-1 YEAR-1
    SEARCH      *CONFERENCE-ON MEETING-1 SUBJECT-1 YEAR-1
    SEARCH      *CONDUCTS-RESEARCH-AT SCIENTIST-2 LAB-1 YEAR-1
EXECUTE?Yes

ENTERING DATA BASE


DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
YES
****************


EVIDENCE CHAIN 1 FROM PLAN 4 PLAUSIBILITY:  99

====================

  **2
  FACT CONFERENCE-ON (IEEE BUBBLE-MEMORIES 1978)
  FACT ATTEND (MACKENZIE IEEE 1978)
  FACT ATTEND (YOSHIDA IEEE 1978)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (YOSHIDA MACKENZIE BUBBLE-MEMORIES 1978)

 **1
 FACT ABOUT (MAG-BUBBLE BUBBLE-MEMORIES)
 ASSUME KNOWS (YOSHIDA MAG-BUBBLE 1978)
 CONCLUDE KNOWS (MACKENZIE MAG-BUBBLE 1978)

**0
FACT CONDUCTS-RESEARCH-AT (MACKENZIE CAMBRIDGE 1978)
CONCLUDE KNOWS (CAMBRIDGE MAG-BUBBLE 1978)

====================
```

Figure 39. Multiple chains of evidence: plan-4 and chain-4

```
MODE: Query:
.((KNOWS)IMP())
(KNOWS  HAS MISSING ARGUMENTS.   3  HAVE BEEN SUPPLIED.)

4 PATHS  11 PROBLEMS  2 PLANS

NEXT?Full plans


<<INFERENCE PLAN 1 PLAUSIBILITY:  99

===================
MAIN FORWARD CHAINS:
--------------------
**1
ASSUME KNOWS (SCIENTIST-2 RESULT-1 YEAR-1)
SEARCH ABOUT (RESULT-1 SUBJECT-1)
CONCLUDE KNOWS (SCIENTIST-1 RESULT-1 YEAR-1)

 **2
 SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-1 LAB-1 YEAR-1)
 CONCLUDE KNOWS (LAB-1 RESULT-1 YEAR-1)

SUPPORTIVE CHAINS:
------------------
 **3
 SEARCH CONFERENCE-ON (MEETING-1 SUBJECT-1 YEAR-1)
 SEARCH ATTEND (SCIENTIST-1 MEETING-1 YEAR-1)
 SEARCH ATTEND (SCIENTIST-2 MEETING-1 YEAR-1)
 CONCLUDE SCIENTIFIC-INFORMATION-FLOW (SCIENTIST-2 SCIENTIST-1 SUBJECT-1 YEAR-1)


 .(()IMP(NOT(KNOWS)))
 (KNOWS  HAS MISSING ARGUMENTS.   3  HAVE BEEN SUPPLIED.)
 DEADEND SUBPROBLEMS THAT REQUIRE NEW PREMISE/TUPLE/PROCEDURE:
 (KNOWS.19.3)
 PARTIAL PLANS?Yes

 4 PATHS  11 PROBLEMS  2 PLANS

 NEXT?Full plans


 <<INFERENCE PLAN 1 PLAUSIBILITY:  99

 ===================

  **2
  SUPP-REQ NOT KNOWS (LAB-1 RESULT-1 YEAR-1)
  SEARCH CONDUCTS-RESEARCH-AT (SCIENTIST-1 LAB-1 YEAR-1)
  CONCLUDE NOT KNOWS (SCIENTIST-1 RESULT-1 YEAR-1)

  **1
  SEARCH CONFERENCE-ON (MEETING-1 SUBJECT-1 YEAR-1)
  SEARCH ATTEND (SCIENTIST-1 MEETING-1 YEAR-1)
  SEARCH ATTEND (SCIENTIST-2 MEETING-1 YEAR-1)
  CONCLUDE SCIENTIFIC-INFORMATION-FLOW (SCIENTIST-2 SCIENTIST-1 SUBJECT-1
 **
 YEAR-1)

 **0
 SEARCH ABOUT (RESULT-1 SUBJECT-1)
 CONCLUDE NOT KNOWS (SCIENTIST-2 RESULT-1 YEAR-1)

 ===================
```

Figure 40. Multiple chains of evidence: Generalized Navigation with KNOWS

```
MODE: Query:

(()IMP(AND(CLOSER-THAN SHIP (KITTYHAWK) PORT)(HOME-PORT (KITTYHAWK) PORT)))

1 PATHS  6 PROBLEMS  1 PLANS

NEXT?Full plans

<<INFERENCE PLAN 1 PLAUSIBILITY:  99

2 SUBPLANS:
===================
**0
COMPUTE GREATER-THAN ((DISTANCE-BETWEEN KITTYHAWK THING-PORT) (DISTANCE-BETWEEN THING-SHIP
THING-PORT))
SEARCH PORTS (THING-PORT)
SEARCH SHIPS (THING-SHIP)
SEARCH SHIPS (KITTYHAWK)
CONCLUDE CLOSER-THAN (THING-SHIP KITTYHAWK THING-PORT)
===================

SEARCH HOME-PORT (KITTYHAWK THING-PORT)

===================


SEARCH/COMPUTE PLAN:
    SEARCH      *SHIPS KITTYHAWK
    SEARCH      *SHIPS THING-SHIP
    SEARCH      *PORTS THING-PORT
    SEARCH      *HOME-PORT KITTYHAWK THING-PORT
    COMPUTE     *GREATER-THAN
                (DISTANCE-BETWEEN KITTYHAWK THING-PORT)
                (DISTANCE-BETWEEN THING-SHIP THING-PORT)

ENTERING DATA BASE

DATA-BASE SEARCH SUCCESSFUL

****************
ANSWER SUMMARY --
VARIABLES:
(SHIP PORT)
ANSWERS:
(GRIDLEY SAN-DIEGO)
(FORRESTAL SAN-DIEGO)
****************

EVIDENCE CHAIN 1 FROM PLAN 1 PLAUSIBILITY:  99

2 CONCLUSIONS:
===================

**0
COMPUTED GREATER-THAN (378 201)
FACT PORTS (SAN-DIEGO)
FACT SHIPS (GRIDLEY)
FACT SHIPS (KITTYHAWK)
CONCLUDE CLOSER-THAN (GRIDLEY KITTYHAWK SAN-DIEGO)

===================


FACT HOME-PORT (KITTYHAWK SAN-DIEGO)

===================
```

3. DESCRIPTION OF THE DADM SYSTEM

3.1 OVERVIEW

The DADM deductive processor (DP) has been designed to interface with existing and emerging relational data management systems (RDMSs). Given this orientation, we have made a sharp distinction between specific facts (n-tuples) which reside in an RDMS data base and general declarative statements (premises) that are directly accessible to the DP. Since the number of general statements that may be required for a practical application is likely to be large (perhaps hundreds or thousands of premises), particular attention has been paid to the development of techniques for the rapid selection of relatively small sets of premises relevant to answering a user's specific request. Premise-selection techniques are automatically invoked when deductive support is necessary to respond to a user's request; otherwise, queries "fall through" the DP and directly drive the RDMS.

This "deductive inference by exception" principle suggests that the DP be viewed as an add-on or enhancement to existing data-base searching capabilities. Such an enhancement can result in a major increase in the power of a data management system by providing a means for extracting and deriving implicit information from data bases of explicit facts.

3.2 APPROACH

Previous approaches to adding deductive capabilities of data management have occurred primarily in the development of question-answering systems (Simmons [14], 15] reviews many of these). The primary deductive methods that have been used are set-inclusion logic, e.g., CONVERSE [2] and SYNTHEX [11]; techniques based on the "resolution" principle [10], e.g., QA3 [1] and MRPPS [9]; procedural-oriented deduction, e.g., SHRDLU [18]; and goal-oriented backward chaining, e.g., MYCIN [16].

A major difference between these systems and our DP is in our use of planning. Our system creates deduction plans to guide the generation of full deductions. We believe such planning to be essential for cutting through the massive number of dead ends and irrelevant inferences which have impaired the performance of earlier systems. Planning becomes even more important for systems involving large numbers of premises. Selection of a manageably small set of possibly relevant premises can be based on such planning.

To this end we have designed and implemented a deductive processor that first builds derivation skeletons which represent possible deduction plans. Once such plans are generated, the system will attempt to instantiate and verify the plans (examine substitutions for variables in premises). We have thus separated the premise-selection process from the process of verifying the consistency of variable substitutions.

The generation of inference plans makes use, when possible, of an efficient technique for <u>middle-term chaining</u> [6]. This process finds implication chains from assumptions to goals through the premises. Middle-term chaining combines the processes of forward chaining from the assumptions in a query and backward chaining from the goals in a query. As chaining proceeds in the two directions, intersections are performed on the derived sets. When a non-empty intersection occurs, the system has found an implication chain from an assumption to a goal. The resulting chain is passed on to the inference plan generator, which extracts the premises whose occurrences are involved in the chain. Subproblems may result, requiring further deduction or data-base search.

A chaining (pathfinding) process does not operate on the premises themselves but on a net structure called the <u>predicate connection graph</u> (PCG). This graph is abstracted from the premises. When a premise is introduced into the system, the deductive connections existing among the predicate (relation) occurrences in the premise are encoded into the PCG. Further, the deductive interations (i.e., unifications [10]) between predicate occurrences in the new premise and predicate occurrences in existing premises are pre-computed

and encoded into the PCG. The variable substitutions required to effect the unifications are stored elsewhere, for latter use by the verifier. Thus, the PCG contains information on the dependencies within premises and the deductive interactions among the premises. During the generation of middle-term chains and plans, the system is aware of the existence of unifications among the premises, but it does not need to generate the unifications nor does it need to examine and combine the variable substitutions associated with the interacting unifications. The former is done by a pre-processor, while the latter is done by the verifier after plans have been generated.

Although some connection graphs used in theorem-proving systems also contain information on the unifications among general assertions (resolution clauses in these systems), they are not used as a planning tool as is the PCG. The PCG most resembles Sickel's clause interconnectivity graph [13] in that both graphs represent the initial deductive search space and are not changed in the course of constructing deductions. Other graph procedures [7, 12] involve adding nodes to graphs as deductions are formed.

3.3 DADM DEDUCTIVE PROCESSOR COMPONENTS

The major DADM Components are illustrated in figure 42. At present users communicate directly with the Controller. At a later date the Controller will communicate with an end user interface such as EUFID The Controller accepts premises, procedural knowledge (as LISP functions), advice rules, queries, and commands. It accesses and coordinates the use of an external RDMS as well as the seven major processing components of DADM:

(1) Array maintenance: This module inserts, deletes, retrives, and compacts (i.e., garbage collects) most forms of information used by the system in LISP arrays. For example, information abstracted from the premises is segmented into seven internal arrays. This segmentation contributes to good system structuring and processing efficiency. Each predicate (relation)

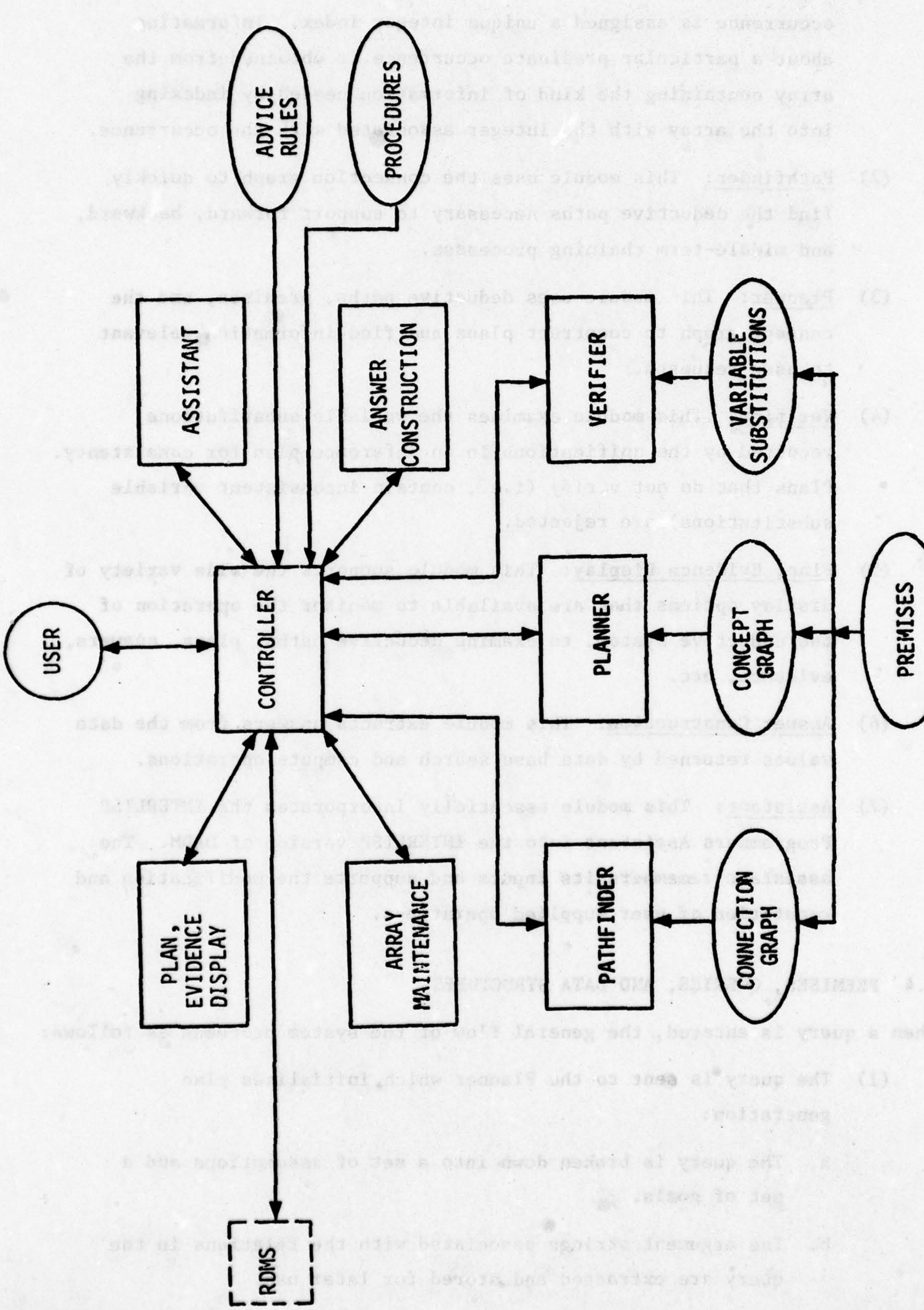


Figure 42. DADM Deductive Processor Components

occurrence is assigned a unique integer index. Information about a particular predicate occurrence is obtained from the array containing the kind of information needed by indexing into the array with the integer associated with the occurrence.

(2) Pathfinder: This module uses the connection graph to quickly find the deductive paths necessary to support forward, backward, and middle-term chaining processes.

(3) Planner: This module uses deductive paths, premises, and the concept graph to construct plans and find information relevant to user requests.

(4) Verifier: This module examines the variable substitutions required by the unifications in an inference plan for consistency. Plans that do not verify (i.e., contain inconsistent variable substitutions) are rejected.

(5) Plan, Evidence Display: This module supports the wide variety of display options that are available to monitor the operation of the deductive system, to examine deductive paths, plans, answers, evidence, etc.

(6) Answer Construction: This module extracts answers from the data values returned by data base search and compute operations.

(7) Assistant: This module essentially incorporates the INTERLISP Programmers Assistant into the INTERLISP version of DADM. The assistant remembers its inputs and supports the modification and repetition of user supplied operations.

3.4 PREMISES, QUERIES, AND DATA STRUCTURES

When a query is entered, the general flow of the system proceeds as follows:

(1) The query is sent to the Planner which initializes plan generation:

a. The query is broken down into a set of assumptions and a set of goals.

b. The argument strings associated with the relations in the query are extracted and stored for later use.

c. A problem graph representing possible inference plans is initialized.

(2) The Pathfinder is called to find chains of middle-term predicate occurrences, via the predicate connection graph, linking assumption predicates to goal predicates. These chains represent attempts to find key predicate occurrences (middle terms) that deductively connect assumptions to goals (via the premises containing the occurrences). Semantic advice in the form of premise and predicate alert lists and the use of variable and constant "types" may also play an important role in the chain generation process.

(3) Using the predicate occurrences within a chain, the Planner extracts the premises containing the given occurrences. The resulting set of premises represents the beginning of an inference plan.

(4) With this set of premises, the Planner examines the predicate occurrences (in the premises) that are not part of the middle-term chain and determines which of these are "unresolved" and need further deductive or data-base support. Each unresolved literal results in the formation of a subproblem and a new node in the problem graph.

(5) An evaluation function examines the nodes in the Problem graph and decides which of these nodes to operate on next. All nodes are considered for selection, those that are subproblems as well as those that are top-level problems (from the input query). Thus, the system may decide to find another middle-term chain for a query goal prior to working on the subproblems resulting from a previously constructed chain. Middle-term chaining continues until all remaining subgoals require data-base support, or until no more chains can be found, or until the chain limit is reached.

(6) After all middle-term chaining is completed, the Verifier attempts to verify the plans in the problem graph. The verifier examines the variable-flow classes of chains comprising each plan to check for inconsistency (no variable taking on two different constant values).

(7) The Data Management System is called for each successfully verified inference plan. The RDMS searches over the data base of specific facts for the remaining subproblems that need data-base support. If data-base search is successful, values for the variables occurring in the search requests are returned and answers are formulated.

REPRESENTATION OF PREMISES AND QUERIES

The basic representation of premises and queries in our system is the primitive conditional. It is a Skolemized, quantifier-free form. However, instead of being a conjunctive normal form as in resolution systems, the primitive conditional retains the implication sign. Primitive conditions have the follwoing possible forms:

(1) &(...)⊃ V(...)

(2) &(...)⊃ &(...)

(3) V(...)⊃ V(...)

(4) V(...)⊃ &(...).

Within the parentheses are literals (negated or positive predicates and their arguments). The primitive-conditional format has the full expressibility of the first-order predicate calculus, i.e., every first-order predicate calculus expression can be represented by one or more primitive conditionals. Note that in resolution, only expressions of type (1) are allowed. They are further modified by transforming the implication into a disjunction with the literals in the antecedent becoming negated, e.g.,

&(A, B)⊃ V(C,D)

is represented as:

$(\neg A \vee \neg B \vee C \vee D)$.

One reason for choosing the primitive-conditional form is to maintain at least some part of the original formulation of an expression as input by a user. Many input expressions map naturally into an implicational form. If they are put into a normal form which does not maintain this implication explicitly, significant clues contained within an expression as to its value for a particular proof or strategy are lost, both to the user and to the system. Furthermore, we want to enable a user interacting with and advising the system to be able to read and understand the evolving inference plans as easily as possible. The use of the primitive-conditional form appears to contribute substantially toward this end.

DATA STRUCTURES

Information abstracted from premises is stored in seven internal arrays. Structural information about the general statements is segmented into four arrays as follows:

(1) The premise array contains all of the general descriptive statements accessible to the system. Each element in the array is a list containing three elements:

a. A list of predicate occurrences in the premise. The occurrences are represented by unique integers which are used to index into the predicate-occurrences array, the arguments array, the unification-arcs array, the variable-substitutions array, and the links array. Information about the structure of premises, argument strings, deductive interactions, etc., are all found in these other arrays.

b. A measure of the plausibility of the premise (for dealing with plausible inference as well as strict inference). Currently, only a very rough measure of plausibility is used.

c. The complete premise in primitive-conditional form. This is for purposes of printout not for analysis and evaluation during the process of deciding whether to use the premise in a possible proof. The information needed for this decision is much more easily available in other arrays.

(2) Each predicate occurrence in the set of premises is given a unique position in the predicate-occurrences array. An entry in this array is a bit vector containing information on the predicate name of the occurrence, the premise which contains the occurrence, the occurrence's numerical position within the premise, whether the occurrence is in the antecedent or consequent of the premise, the connective under whose scope the occurrence lies, and the sign of the occurrence.

(3) The argument string of each predicate occurrence is stored in the arguments array in the position corresponding to the integer index assigned to the occurrence.

(4) Every predicate name occurring in the premises is stored in the predicates array. (Predicate names should not be confused with predicate occurrences which are particular instances of predicate names within the premise set. For example, SCIENTIST is a predicate, but in a premise referring to Einstein as a scientist, the particular occurrence of the predicate, SCIENTIST (Einstein), must be identified and distinguished from the predicate in general.) Each predicate has a property list containing the indices of all occurrences of that predicate in the premise set.

Possible deductive interactions between expressions exist as "unifications" as described by Robinson in his development of the resolution principle [10]. Unification is a matching procedure that finds necessary substitutions for variables in order to effect deductive interactions. For example, if we know that Joe is a man, i.e., MAN(Joe), and that all men are human, i.e.,

$\forall_x$(MAN(x) $\supset$ HUMAN(x)),

then we can conclude that Joe is human, i.e., HUMAN(Joe). The unification procedure determines that the substitution Joe for the variable x is needed in order to make the desired conclusion. In most resolution-type inference systems, procedures to detect and compute unifications are executed repeatedly. In contrast, our deductive processor pre-computes all possible unifications that exist among premises and stores them.* This is done when premises are first introduced into the system. The inference planning process uses the information about the existence of unifications but is not charged with the formation of them. Once inference plans have been formed, the Verifier examines the unifications within the plan to determine if there are any variable substitution conflicts.

Two internal arrays store information about unifications:

(5) For each predicate occurrence, a list of the indices of the predicate occurrences that unify with it are stored in the unification-arcs array in the entry corresponding to the index of the occurrence.

(6) The variable-substitutions array stores the substitution lists associated with the unifications in a one-to-one correspondence with the entry of unifications in the unification-arcs array. Substitution lists specify varibles and constants that must be made identical for unifications to take place.

The final array contains information on the predicate dependencies of occurrences within premises. This "links" array will be discussed in the predicate connection graph description.

*In resolution jargon, this would be stated as computing all possible unifications that exist among original clauses.

3.5 DEDUCTIVE PATHFINDING

MIDDLE-TERM CHAINS

The concept of a middle-term chain is central to the operation of the DADM inference system. Syntactically, a chain is a list of predicate occurrences. A given input query contains a set of assumptions and a set of goals. The first element in a middle-term chain is an occurrence, within the premise set, that unifies with an assumption predicate. The last element in a chain is an occurrence, within the premise set, that unifies with a goal predicate. A goal predicate is either a query goal or an internally generated subgoal. The other elements in the chain are produced by the chain generator as it alternately finds links and unification-arcs (u-arcs). Links connect occurrences within premises while u-arcs connect premises with one another through predicate occurrences that deductively interact.

Consider, for example, the query "$A \supset D$?" and suppose the following premises were known to the system (argument strings have been suppressed for simplicity):

(1) $A_1 \supset B_1$

(2) $B_2 \supset C_2$

(3) $C_3 \supset D_3$

The subscripts serve to distinguish the predicate occurrences by identifying the premises in which they occur. Unifications might exist between B_1 and B_2 and between C_2 and C_3. Links (discussed shortly) exist between A_1 and B_2, between B_2 and C_2, and between C_3 and D_3. From the query, the assumption predicate is A, and the goal predicate is D. Within the premise set we find A_1, an occurrence of the assumption predicate A, and D, an occurrence of the goal predicate D. Assume unifications exist between A and A_1 and between D and D_3. The chain generator would produce the chain:

$$A_1 \underbrace{\qquad}_{\text{link}} B_1 \underbrace{\qquad}_{\text{u-arc}} B_2 \underbrace{\qquad}_{\text{link}} C_2 \underbrace{\qquad}_{\text{u-arc}} C_3 \underbrace{\qquad}_{\text{link}} D_3$$

The predicates B and C are considered "middle-term" predicates, i.e., predicates that are needed to link the assumption A to the goal D. The predicate occurrences B_1, B_2, C_2 and C_3 are middle-term predicate occurrences. A_1 and D_3, the chain end points a, are occurrences of the assumption and goal, respectively.

Some earlier researches in the field of mechanized inference have built deductive mechanisms that rely primarily on generating implication chains. We have extended these earlier efforts by allowing more complex premises, by specifying and using different types of logical dependency among concepts, by combining these with unifications of predicates between premises, and by using the chains not as final products but as a means of generating derivation plans for a general-purpose inference system.

CONNECTION GRAPH

The predicate connection graph is contained within two arrays, the links array and the unification-arcs (u-arcs) array. The u-arcs array contains the unifications that exist within the premise set. All possible unifications among the premises are pre-computed and stored. The links array contains information about predicate occurrences as they relate to each other within a premise. The information involves the concepts of dependency and linkages which are discussed in this section. These two arrays are used by the Pathfinder to generate middle-term chains.

PREDICATE OCCURRENCE DEPENDENCIES

The concept of dependency discussed here involves the relationship between predicate occurrences within a particular premise. A predicate occurrence in a particular premise can be, but is not necessarily, truth-functionally dependent on other predicate occurrences in the same premise. Dependency does not extend across premises. It is the unification-arcs that are involved in premise to premise interaction.

A premise is considered to be indivisible if it cannot be broken up into two or more disjoint premises. Two predicate occurrences are dependent on one another if they occur within an indivisible premise.

Consider the premise

(a) $v(P, Q) \supset R$.

It can be divided into two distinct indivisible premises which are logically equivalent to the original premise when conjoined, namely,

$P \supset R$ and $Q \supset R$.

Thus, P and R are dependent on each other as are Q and R. However, P and Q are independent even though they both occur within the same original premise. A similar situation arises in the premise

(b) $R \supset \&(P, Q)$.

Once again there is no dependency between P and Q. Such is not the case in the premises

(c) $\&(P, Q) \supset R$ and

(d) $R \supset v(P, Q)$.

These premises cannot be subdivided and are thus indivisible. Dependencies exist between P and Q in both cases as do the other two dependencies (between P and R and between Q and R).

The procedure for identifying dependencies among predicate occurrences in premises is straightforward given the primitive-conditional form for premises. Predicate occurrences within a disjunction on the lefthand side of an implication are not dependent on each other, nor are predicate occurrences within a conjunction on the righthand side. All other predicate occurrence pairs within a premise entail a dependency.

LINK TYPES

Four types of links are used to represent dependencies between predicate occurrences within a premise represented in the primitive-conditional format.

(1) Implication (I) Links

This type of link can be represented in its simplest form as:

$$A \supset B$$

or, more generally, as

a. $c_1(\ldots A \ldots) \supset c_2(\ldots B \ldots)$.

where c_1 and c_2 can be either of the two connectives "&" or "v" (as will be the case in all subsequent examples). The dots represent either the empty expression or other atomic components of the antecedent or of the consequent. It is to be understood that the predicates shown do not fall within the scope of any negation sign not explicitly shown. Type I links are asymmetric and are referred to as a link <u>from</u> A <u>to</u> B in the above examples.

Type I links also exist from A to B in the following expressions:

b. $\&(\ldots A, \neg B \ldots) \supset c_1(\ldots)$

c. $c_1(\ldots) \supset v(\ldots \neg A, B \ldots)$

d. $c_1(\ldots \neg B \ldots) \supset c_2(\ldots \neg A \ldots)$.

Note that the main connective in the antecedent in b. must be & for a link to exist between A and B; otherwise A and B would be independent. Similarly, the connective in the consequent of c. must be v.

(2) Reverse Implication (RI) Links

Whenever a type I link exists from one predicate occurrence to another, as from A to B in the above examples, a type RI link exists in the opposite direction, <u>from</u> B <u>to</u> A. Such links are needed because of the one-directional aspect of the type I link. The addition of the RI link enables the predicate connection graph to be traversed both <u>from</u> and <u>to</u> any given predicate occurrence. Looking at the examples above for I links from A to B, we note that in all cases an RI link exists from B to A. The

simplest form of the RI link (from B and A) derives from

$\neg B \supset \neg A$.

(3) Conjunction (C) Links

The basic primitive-conditional form in which a C link occurs (between occurrences A and B) is

$\&(\ldots A, B \ldots) \supset c_1(\ldots)$.

Other representations in which C links occur include

$c_1(\ldots A \ldots) \supset c_2(\ldots \neg B \ldots)$,

which is its simplest form is

$A \supset \neg B$,

and

$c_1(\ldots) \supset v(\ldots \neg A, \neg B \ldots)$.

Type C links are symmetric in that if A is linked to B by a type C link, so is B to A.

(4) Disjunction (D) Links

The basic primitive-conditional form in which a D link occurs (between occurrences A and B) is

$c_1(\ldots) \supset v(\ldots A, B \ldots)$.

Other representations in which D links occur include

$c_1(\ldots \neg A \ldots) \supset c_2(\ldots B \ldots)$,

which in its simplest form is

$\neg A \supset B$,

and

$\&(\ldots \neg A, \neg B \ldots) \supset c_1(\ldots)$.

Type D links are also symmetric.

The links array contains information on all of the links within the set of premises. Indexing into this array is similar to the indexing into the other arrays. The unique integer identifying a predicate occurrence in the premise set is used to index into the links array in which is entered, for each occurrence, the dependency links which emanate from it. Each entry is a list of four sublists: the I-Linked, the C-linked, the D-linked, and the RI-linked predicate occurrences.

Specifications of link types provides an efficient means of storing information about predicate occurrence dependencies and greatly facilitates the chain generation process. Figure 43 lists link restrictions that must occur within chains in order to effect logical validity. Row 1 indicates that if an assumption predicate is positive, the first link in a chain must be of type I or C. For example, if A is an assumption, a link such as one found in the premises $A \supset B$ (type I), $A \supset \neg B$ (type C), $\&(A, B) \supset D$ (type C between occurrences A and B), etc., could be used in initializing a chain. The MTCG would thus examine the links array for links of type I or C out of occurrences of the predicate A. If the assumption were negative ($\neg A$), the MTCG would locate RI and D links.

If the goal predicate is positive (row 3 in Figure 43), the last link in a chain must be of type I or D. For example, if G is a goal, the last link in a chain could be found in premises such as $E \supset G$ (type I), $\neg E \supset G$ (type D), etc. Note that in actual operation, the Pathfinder would be working backward from the goal and would be looking for an RI or D link out of G (which would result in an I or D link into G).

	If assumption predicate is	First link in chain must be of type
(1)	+	I or C
(2)	-	RI or D

	If goal prediate is	Last link in chain must be of type
(3)	+	I or D
(4)	-	RI or C

	From link of type	The link following it must be of type
(5)	I	I or C
(6)	RI	RI or D
(7)	C	RI or D
(8)	D	I or C

Figure 43. Link Restrictions within Chains.

Rows 5 through 8 in Figure 43 list the restrictions of what link types may directly follow other link types in a middle-term chain. For example, given a type I link in a chain, as in $A \supset B$, the next link must be of type I, as in $B \supset C$, or of type C, as in $B \supset \neg C$. Given a type RI link in a chain, as in $\neg D \supset \neg E$, the next link must be of type RI, as in $\neg E \supset \neg F$, or of type D, as in $\neg E \supset F$. The restrictions on successive links apply for finding links out of both assumptions and goals. As an example of the latter case, consider the goal $\neg G$. The last link in a chain <u>to</u> this goal must be of type RI or C (row 4). The Pathfinder thus looks for links of type I or C <u>out</u> of G. Suppose it picks up the C link between E and G in $E \supset \neg G$. Now, the Pathfinder must find links of type RI or D <u>out</u> of E (resulting in links of type I or D <u>into</u> E). Note that this is precisely the restriction specified in row 7. Thus, the restrictions in rows 5 and 8 apply to links out of assumptions and to links <u>out</u> of goals

SEMANTIC INFORMATION

Semantic Advice

A data-base administrator may enter semantic advice in the form of "Condition $\Longrightarrow$ Recommendation" rules. For example, one could advise that a ship return to its home port if it is damaged by specifying:

(ASSUMPTION: DAMAGED(SHIP)) $\Longrightarrow$ RETURNS(SHIP PORT)

The system would try using premises containing the RETURNS relation when the DAMAGED relation occurs as an assumption. Advice rules are stored in an advice array, where they are automatically selected and applied whenever their condition part holds for input queries. In addition to such advice rules, the user may supply advice for a particular query by stating only the advised recommendation for that query.

Advice most typically involves recommendations on the use of particular premises or predicates in finding deductions. For advised premises, the system will try using them whenever possible in the course of constructing a proof. For advised predicates, the system will try chaining through occurrences of them in premises. In the case of negative advice, specified premises and predicates are avoided in plan construction.

Advised premises and predicates are placed on the premise and predicate alert lists. These lists are used in two ways. During the chain construction process, the Pathfinder considers several possible predicate occurrences in its search for links and u-arcs. Those occurrences that represent instances of advised predicates or that occur within advised premises are given preferential status in chain generation. In addition, completed chains for query goals are examined and only those chains having premises or predicates that occur on one of the alert lists are passed on to the Planner (Chains that are formed for subgoals need not pass this test since the subgoals resulsted from chains which did use advice.) Advice is thus used both for pruning within chain generation and as a basis for evaluatively filtering chains.

Advice given by a user might be based on his knowledge of the domain, concepts or predicates most frequently used in plans, premises that have previously been successful in plans, and intuition (which should not be underestimated). Also, the user may direct the system to use a particular proof strategy by advising the use of a particular premise, e.g., the premise v(X, Y) for a proof-by-cases strategy. If no usable inference plans are developed from some given advice, the user may re-input (redo) the query with different or no advice.

Variable and Constant Types

When entering a premise or query into the system, the user may specify a class membership "type" for any variable or constant in the expression. Class membership is typically specified by one-place relations in predicate calculus representations. For example, to specify that a variable x ranges over scientists, one enters an expression such as SCIENTIST(x). Similarly for constants, as in SCIENTIST(Einstein). We have allowed the specification of these membership constraints within premises and queries without the need for these one-place relations.

Compound types, consisting of set union, intersection, and difference operations over simple types, may also be used to specify more complex semantic restrictions on predicate domains. The Concept Graph is used to represent set relationships between types. Class inclusion paths within this network are used, for example, to permit unification of instances of type SCIENTIST with instances of type MAMMAL. As new premises are entered into the system, this semantic network is automatically updated to reflect new predicate-domain associations.

The use of such semantic information aids the deductive process in three ways. First, premises and queries may have fewer relations by the elimination of some one-place relations. This results in fewer goals and subproblems to solve because of fewer unresolved literals. The size of the problem graph would correspondingly be reduced.

Secondly, there is a reduction of the storage space required for these one-place relations within the various arrays of information. It is possible to eliminate predicates, such as SCIENTIST, and occurrences of these predicates in the premise set. This results in the elimination of links and unifications for such occurrences.

Thirdly, the number of possible unifications among the remaining occurrences in the premises is reduced. There is also a reduction in the number of unifications beteen query predicates and premise occurrences. We have modified the unification algorithm to check for variable and constant types as it matches argument strings. Added to unification is the constraint that two arguments being matched must be of the same type or one argument must be typeless. The reduction of unifications enhances the operation of the system, since it has less unifications to consider within the chaining process.

3.6 GENERATION OF INFERENCE PLANS

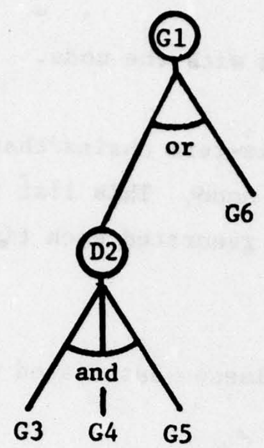
PROBLEM GRAPH

Structure

Inference plans are stored in a problem graph. Nodes in the graph are created initially from the input query when the assumption and goal sets are extracted. Later, during the derivation planning process, the generation of middle-term chains often results in the creation of subproblems from the unresolved literals in premises associated with the chains. New nodes are created for these subproblems.

Two types of nodes are distinguished in the problem graph. "Goal nodes" contain query goals or system generated subgoals that need to be established. The Planner sends the information residing in one of these goal nodes to the Pathfinder which returns a middle-term chain. The Planner determines the subproblems associated with the chain and creates goal nodes for them. The nodes hang from the particular node that was used for creating the chain. The Planner may later decide once again to use the same node in calling the Pathfinder to find an alternative chain. The subproblems resulting from this new chain also hang from the same goal node. Thus, a goal node may have several branches hanging below it, one branch for each chain and the set of subproblems resulting from it. The branches are implicitly disjoined, i.e., each branch is one possible derivation of the goal and only one of them need be considered for a particular proof.

Since each branch represents one middle-term chain and a set of subgoals, the need for a second type of node, the "dummy node", arises. Dummy nodes serve to specify a set of conjunctive elements (conjoint subproblems) within a disjunctive set of branches (alternative chains). (Dummy nodes are also used to specify a set of disjunctive elements falling under a conjunctive set of branches (alternative chains). (Dummy nodes are also used to specify a set of disjunctive elements falling under a conjunctive set of branches.) Consider the problem graph.



(Nodes are labeled for reference purposes and do not show the information contained within them.) G1 is a goal node containing a set of assumptions and a goal from the input query. The branch to the dummy node D2 results from a middle-term chain derived *from* the information in G1. The three subproblems, goal nodes G3, G4, and G5, hanging from D2, are created from the unresolved liaterals in the premises containing the links of the middle-term chain. These subproblems must all be solved if the branch to D2 is used in a proof. The branch to G6 is formed from an alternative middle-term chain. In this case, however, only one subproblem is formed and it is contained in G6. A dummy node is not needed.

Each dummy node created by the system has a property list consisting of two elements: the node's successors (G3, G4, and G5 for the node D2 above), either AND'd or OR'd, and the node's parent (G1 for D2 above).

Each goal node also has a property list which contains the following elements:

(1) The successors out of the node, always implictly OR'd. One successor is created for each chain that was generated from this node. (When a node is "closed", indicating no subgoals result from a chain, or when a node contains a goal that is to be resolved via the fact file, an integer flag is placed in this position.)

(2) The assumptions associated with the node.

(3) The goal associated with the node.

(4) A list of the middle-term chains that have already been generated for this node. This list is needed so duplicate chains will not be generated each time the node is used for chain generation.

(5) The verification classes associated with the chain that formed this node.

(6) The node's parent.

Items 1 and 6 and the information on the property lists of the dummy nodes determine the structure of the problem graph. Items 2, 3, and 4 are used by the Pathfinder for chain generation. Item 5 is used during verification.

Initialization

A query in the primitive-conditional form is input by a user. The antecedent (left-hand side of the implication) and the consequent (right-hand side of the implication) are extracted from the query. The predicates in the antecedent are considered assumptions. Those in the consequent are considered goals. If the main connective in the antecedent is AND, the predicates under its scope are included in the set of assumptions for each of the goals in the consequent (examples 2 and 3 in Figure 44). Any or all of the assumptions may be used in deducing the goals. If the main connective is OR, a conjunctive dummy node (e.g., D1) is created in the tree such that each predicate in the antecedent is treated individually with respect to the goals, i.e., a subproblem is created for each assumption predicate with respect to each goal predicate. All of the subproblems would need to be established. This can be seen in the sixth example query in Figure 44. The query is equivalent to:

$((A \supset C) \& (B \supset C))$.

To establish it, we need to show that A implies C and that B implies C.

Within the consequent, if the main connective is AND, each predicate is considered a goal to be deduced from the assumption set. A conjunctive dummy node is created, all of whose goals must be established (example 3 in Figure 44). If the main connective is OR, a disjunctive dummy node is created where each consequent predicate is a goal, but only one of the specified goals need be established (examples 5 and 7 in Figure 44). The assumption set for each goal also includes the negation of the other consequent predicates. This results from the equivalence of:

$A \supset v(B, C)$, $\&(A, \neg B) \supset C$, and $\&(A, \neg C) \supset B$.

Thus, if the first expression were input as a query, two disjoined goals would be formed, one with an assumption set $(A, \neg B)$ and a goal set (C), the other with an assumption set $(A, \neg C)$ and a goal set (B).

Other examples queries and their corresponding initial problem graphs are also shown in figure 44. In particular, the query in example 4 contains no assumptions. Example 8 gives an example of the most complicated type of initial problem graph, i.e., one for a primitive-conditional query having a disjunctive antecedent and a disjunctive consequent.

NODE EVALUATION AND SELECTION

Given a problem graph, the Planner must decide which of the goal nodes should be used in calling the Pathfinder, i.e., which subproblem to work on next. Two measures used for determining this selection are: the age of the node and the number of subproblems.

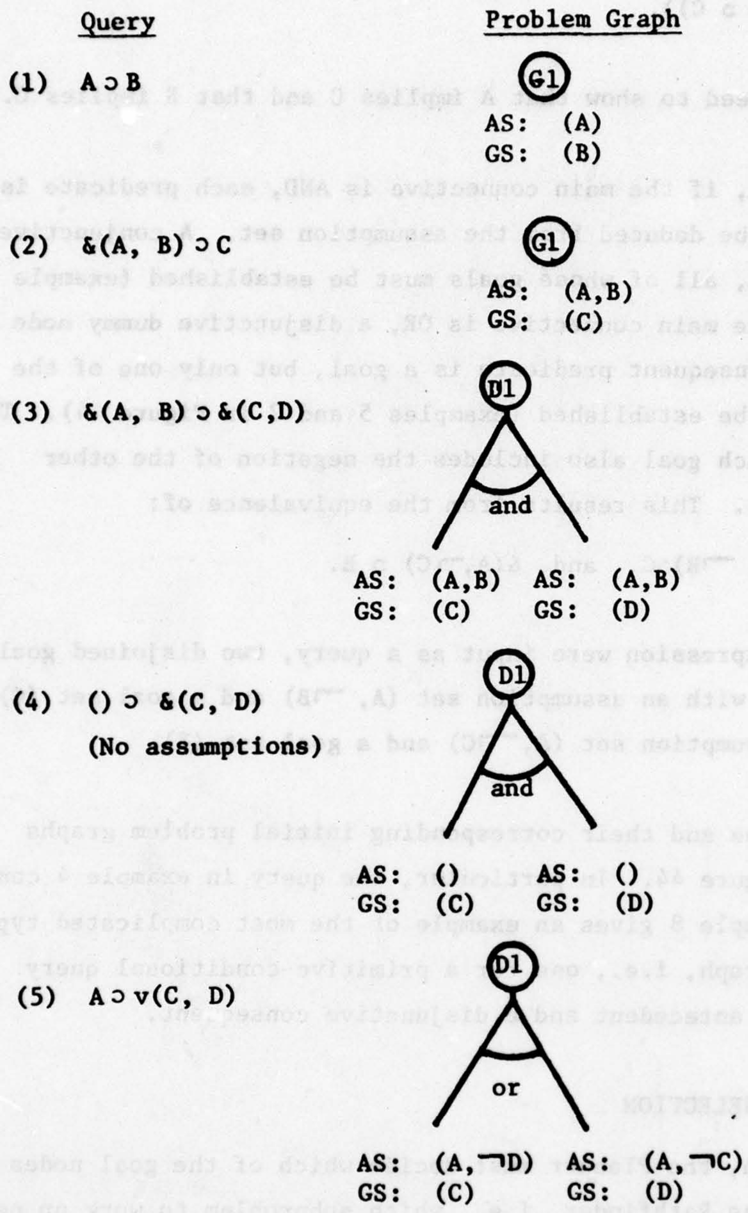
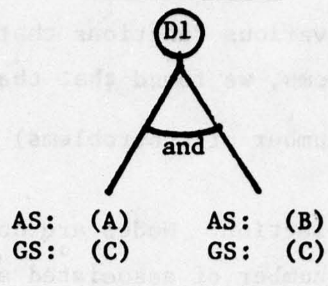


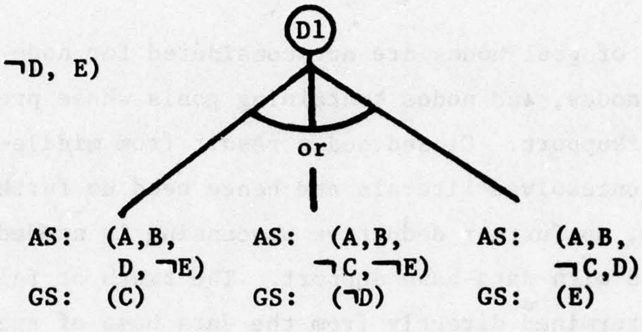
Figure 44. Queries and their Initial Problem Graph (1 of 2) (AS: indicates Assumption Set; GS: indicates Goal Set)

Query **Problem Graph**

(6) $\vee(A, B) \supset C$



(7) $\&(A, B) \supset \vee(C, \neg D, E)$



(8) $\vee(\neg A, B) \supset \vee(C, \neg D)$

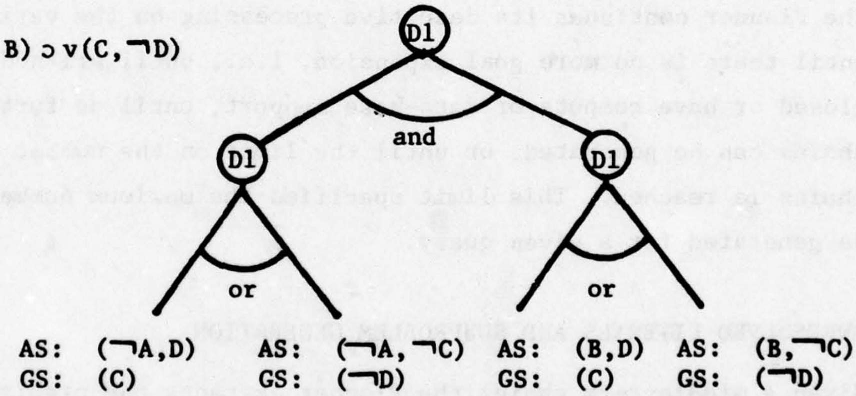


Figure 44. Queries and their Initial Problem Graph (2 of 2) (AS: indicates Assumption Set, GS: indicates Goal Set)

In experimenting with various functions that combine the measures of age and number of subproblems, we found that the evaluation function:

(2 x age) - (number of subproblems)

serves well in node selection. Nodes are not allowed to become too old, and nodes with a high number of associated subproblems are temporarily set aside but are not forgotten.

Two types of goal nodes are not considered for node selection: closed or terminal nodes, and nodes containing goals whose predicates have complete data-base support. Closed nodes result from middle-term chains that yield no unresolved literals and hence need no further deductive processing. Similarly, no further deductive processing is needed for goals containing predicates with data-base support. The truth or falsehood of these goals can be determined directly from the data base of specific facts.

The Planner continues its deductive processing on the various goal nodes until there is no more goal expansion, i.e., until all nodes are either closed or have compute or data-base support, until no further middle-term chains can be generated, or until the limit on the number of middle-term chains is reached. This limit specifies the maximum number of chains to be generated for a given query.

UNRESOLVED LITERALS AND SUBPROBLEM GENERATION

Given a middle-term chain, the Planner extracts the premises containing the links of the chain. For each link, the premise containing it is examined to determine which of the literals in the premise are unresolved, i.e., those needing further deductive or data-base support. These unresolved literals result in the creation of subproblems.

The notion of predicate occurrence dependency, discussed earlier, can be used to determine which of the literals in a premise are unresolved. Dependency between any two occurrences in a premise is based on the side of the implication sign on which the occurrences reside and on the main connective governing the occurrences. (The positive or negative signs of literals are needed to determine link types between occurrences but are not needed for establishing dependency). Whether an occurrence in a particular premise is unresolved is based on its relationship to the two premise occurrences involved in a link in a middle-term chain.

A literal in a premise is unresolved if it is dependent on both of the two linked occurrences in the premise. The literal must be dependent on both; otherwise the premise can be subdivided such that the literal does not occur in the sub-expression involving both linked occurrences. The sub-expression containing the link is the one needed in the derivation. A literal not occurring in this sub-expression need not be considered in the proof. For example, suppose a chain included the link between A and C in the premise

$$\&(A, B) \supset \&(C, D).$$

The occurence B is dependent on both A and C and is therefore unresolved The occurrence D is dependent on A but independent of C and is therefore not unresolved. Dividing the premise into

$$\&(A, B) \supset C \quad \text{and} \quad \&(A, B) \supset D,$$

we note that the first expression includes the link between A and C and is therefore the one required for the derivation using this link in a middle-term chain. This expression does not contain D.

Once the Planner has determined which literals in a premise are unresolved, it must then examine the dependencies among the set of unresolved literals. Consider the premise:

$$\&(A, B) \supset \&(C, D),$$

and suppose the link between A and B occurs in a middle-term chain. Occurrence C is dependent on both A and B and is thus unresolved. The same is true for D. The premise, however, is not indivisible since it can be divided into the two expressions

$$\&(A, B) \supset C \quad \text{and} \quad \&(A, B) \supset D.$$

Both expressions contain the link between A and B reconfirming that both C and D are unresolved. Because these expressions are premises in themselves (resulting from the original premise), only one of them need exist in a derivation involving the link between A and B. Thus, the Planner need resolve C or D but need not resolve both. This is due to the independence of C and D. The subproblems would result in the creation of a disjunctive branch in the problem graph. In the more general case, those occurrences in the unresolved set that are independent of one another will fall under a disjunctive branch; those that are dependent will fall under a conjunctive branch where all the unresolved occurrences must be resolved

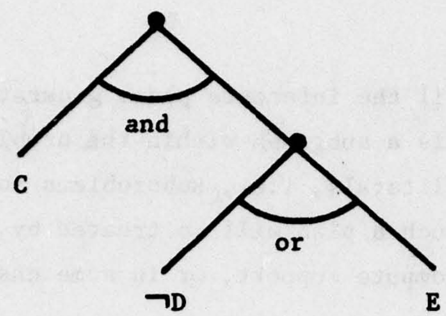
The Planner creates a subproblem for each unresolved literal. If the literal occurs in the antecedent of a premise it remains unchanged as it becomes a subgoal. If the literal occurs in the consequent, however, the literal is negated when it becomes a subgoal. This is done so that when the unresolved literals are established, they will correctly unify with literals in other premises in the proof.

Consider a link between A and B in the premise

(a) $\&(A, B, C) \supset \&(D, \neg E)$.

Occurrences C, D, and E are unresolved with D and E independent. The following branch would be added to the problem graph under the node from which a middle-term chain involving the premise was formed (only the goal is shown for each node).

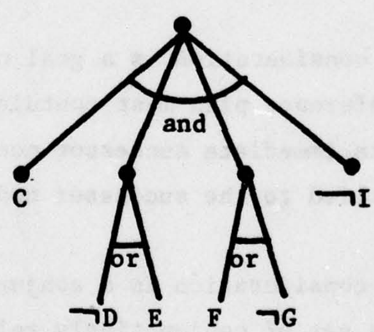
(b)



This branch is conjoined with other branches resulting from unresolved literals in other premises associated with the chain. Suppose the link between B and H in the premise:

$$v(F, \neg G) \supset v(B, H, I)$$

occurs in the same middle-term chain as the link in premise (a) above with a u-arc existing between the B occurrences. Occurrences F, G, and I are unresolved *with F and G independent.* The following branch would be created for the two premises:



EXTRACTING INFERENCE PLANS

The problem graph contains all the inference plans generated by the system. A particular inference plan is a subgraph within the problem graph. A plan may contain unresolved literals, i.e., subproblems not yet resolved. The unresolved literals in such a plan will be treated by the system as needing data-base support, compute support, or in some cases (partial plans) deductive support.

The following set of rules specify the requirements for extracting an inference plan from the problem graph. One begins by applying the rules to the top-level node in the graph.

(1) If the node under consideration is a goal node and the node is closed (no unresolved literal exists in the node), the inference plan must contain the node.

(2) If the node under consideration is a goal node and the node has an unresolved literal which needs data-base or compute support, the inference plan must contain the node.

(3) If the node under consideration is a goal node and the node has successors, the inference plan must contain the node and one and only one of its immediate successor nodes*. The five rules are then applied to the successor node chosen.

(4) If the node under consideration is a conjunctive dummy node (indicating that a set of conjunctively related branches hang below), all of the immediate successor nodes must be part of the inference plan. The five rules are then applied to each of the immediate successors.

*Recall that the set of branches hanging from a goal node is disjoined, one branch for each middle-term chain generated for the node.

(5) If the node under consideration is a disjunctive dummy node (indicating that a set of disjunctively related branches hang below), one and only one of the immediate successor nodes must be part of the inference plan. The five rules are then applied to the immediate successor node selected.

3.7 VERIFICATION OF INFERENCE PLANS

A unification, or deductive interaction, between two predicate occurrences of the same relation name has a substitution list associated with it. This list specifies what substitutions for the variables in the occurrences are needed to make the argument strings of the occurrences identical. A typical inference plan involves several unifications. The primary function of the verifier is to examine the substitution lists of the unifications and check for substitution consistency.

The procedure used in the verifier examine variable flows within the unifications and combine variables and constants that must be equal into variable flow or verification classes. Whenever a variable-flow class has two or more difference constants in it, the inference plan is "blocked" and verification fails. Each variable in an inference plan can take on at most one constant value.

As an example, consider the unification of the two literals A(a,x,y) and A(x,b,w), where "a" and "b" are constants and "w", "x", "y", are "z" are variables. The substitution list for this unification is (a/z, b/x, y/w), which reads "a" substituted for "z", "b" substituted for "x", etc. Also consider the unification of the literals B(w,z) and B(c,v), where "c" is a constant and "v" is a variable. The substitution list is (c/w, v/z). Now suppose both of these unifications occur within an inference plan, such as in:

```
... ⊃A(a,x,y)
      1 |
      A(z,b,w) ⊃B(w,z)
                2|
                B(c,v,)⊃ ...
```

The variable "z" must be identical to the constant "a" according to unification 1, and identical to the variable "v" according to unification 2. Combining these unifications within the proof, we obtain the variable-flow class (a, z, v). Other variable-flow classes for the above example are (b, x), according to unification 1, and (c, w, y), according to both unifications.

The variable-flow classes serve to monitor variable substitutions within a middle-term chain and within a set of chains comprising an inference plan. When a variable is required to be substituted by two difference constants, a blockage results. If this occurs in a chain, no further planning will involve the chain. If a blockage occurs in an extracted inference plan, the plan fails and data-base search requests are not formed for the remaining subproblems.

One other type of blockage can occur during verification. In combining classes within verification, the verifier must examine the variable and constant "types" of the elements within the classes. If an element has a specific type, other elements in the same class must have the same type or be typeless. Otherwise a blockage occurs.

3.8 DATA BASE SEARCH

A given inference plan may have remaining subproblems that need data-base support, i.e, support from the file of specific facts. These remaining subproblems are set up in the form of search requests for the Data Management System (RDMS). The RDMS, in turn, searches the data base to find facts that are instances of these search requests. If all search requests are satisfied, the inference plan becomes a complete proof and answers can be generated. If data-base search fails, the inference plan is unsuccessful.

One important mechanism the RDMS has is the ability to generate conditional answers. This will occur under certain circumstances if RDMS search is partly successful and there is insufficient information available for the remaining search requests. The RDMS can then notify the user that certain specific information is needed to complete the inference plan. Thus the system can be utilized, in some cases, to tell the user what facts are needed to answer his query.

To search an external (i.e., non-LISP) data base each relation associated with the data base must be marked as EXTERNAL and have data base field names supplied through use of the adjust mode. Then if Control mode: IL is turned on relational queries in an Intermoderate Language format will be printed out at the user's terminal and also be sent to a disk file for transfer to the external data base system.

3.9 RECURSIVE PREMISES AND SPECIAL PURPOSE GENERATORS

Premise of the general form:

$$P(\_\,\_\,\_)\ \&\ \_\,\_\,\_\,\_\,\_\ \supset\ \_\,\_\,\_\,\_\ P(\_\,\_\,\_)$$

are recursive and can lead in some circumstances to excessive growth of search space in deductive systems. In addition to advice and variable typing DADM uses a third technique to reduce the problems caused by recursive premises.

Unification between multiple occurrences of a predicate within the same premise may often be avoided by restating the premise's assertion by use of logical properties.* For example, the predicate "North-of" could be characterized by the premises:

$$VxVy\ (\text{North-of}(x,y)\ \&\text{North-of}(y,z) \supset \text{North-of}(x,z))$$
$$VxVy\ (\text{North-of}(x,y) \supset \text{North-of}(y,x))$$
$$Vx\ (\ \text{North-of}(x,y))$$

The first premise specifies that North-of is transitive. This premise can deductively interact with itself and the other premises to cause a rapid expansion of the deductive search space. To help avoid this problem, DADM permits binary predicates to be characterized by their logical properties (for example North-of would be assigned the logical properties: transitive, asymmetric, and irreflexive). Generators can then be called to effect special-purpose inferences associated with various groupings of logical properties. Recursive premises describing logical properties of predicates are therefore replaced, where possible, by special-purpose subroutines.

Logical properties of binary relations are identified by a user-system dialog illustrated below, for the predicate "North-of" (user input is preceded by an asterisk):

* Define (North-of)

Suppose one thing is North-of a second thing that in turn is North-of a third thing. Is the first thing North-of the third?

* Yes

If one thing is North-of a second thing, will it always be the case that the second is North-of the first?

* No

---

*Examples are:
reflexive (equal-to), irreflexive (greater-than), symmetric (equal-to), asymmetric (North-of), transitive (located-in), 1-leader (mother-of), 1-follower (weighs), noregrowth (son-of), and unlooped (mother-of).

Might it ever be the case?

* No

After the third yes/no response, the system is able to identify "North-of" as a transitive, asymmetric, irreflexive, and unlooped relation.

In *one use of this technique* a series of recursive premises were replaced by an equivalence class generator. A proof that had required eleven premise statements was reduced to one containing only five premises plus the equivalence class generator. Similar savings appear to be possible in many other recursive premise situations.

3.10 DADM PRINT AND CONTROL MODES

DADM can run in several different control modes and can printout or display a wide range of information about paths, plans, verification classes, answers, evidence, etc. The following print and control modes can be easily set by the use of the adjust mode:

<u>DADM PRINT MODES</u>

PADVICE	Print advice alert lists
PPATH	Print all middle-term chains
PMAIN	Print main chain paths only
PPATHO	Print occurrence indices for each chain printed
PEFFORT	Print effort indicators for each chain printed
PPREM	Print premises for each chain printed
PVERC	Print verification classes for each chain printed
PSUBG	Print subgoals for each chain printed
PVERP	Print resultant classes for verified plans and final classes for each successful data-base search
PSR	Print search requests and compute relations for each verified plan
PDV	Print data values for each successful data-base search

DADM PRINT MODES (cont'd)

PISR	Print instantiated search requests for each successful data-base search
PANSWER	Print answer information for each successful data-base search
PDVALL	Print summary of data values found during data-base search
PANSALL	Print answer summary
PROOFA	Automatic proof display
PROOFM	Manual proof display
PPLAN	Print inference plans (includes PSR)
PSENT	Print plans, proofs in external format
PDIS	Print plans, proofs in internal format
PLANREPT	Print plans using same premises as previous plans

DADM CONTROL MODES

NODMS	No data management search
VER1	Verify oneplan at a time
DMS1	One data-base search at a time
AQ	Automatic query when entering DERIVE (), or DADM()
NOVER	No verification of plans
IL	Generate and print IL search requests

4. SPECIFIC TASKS ACCOMPLISHED

During the period of performance (1 April 1976 to 30 December 1978) we have accomplished the following tasks:

(a) Implemented the DADM prototype in SDC LISP 1.5 on the IBM 370/158 and AMDAHL470/V6. This prototype consists of the following modules: controller, array maintenance, pathfinder, planner, verifier, plan-evidence display, and answer construction.

(b) Converted the DADM prototype for operation in INTERLISP under TENEX for use on DEC-10 computers. As part of this process restructured LISP code, converted to INTERLISP FOR macros and CLISP.

(c) Added a user assistant module to INTERLISP version of DADM.

(d) Implemented ability for DADM to control and access local RDMS (in LISP) as well as remote DMS (via Intermediate Lanaguage queries).

(e) Implemented an extensive series of user prompt, guidance, help, and break (interrupt) facilities.

(f) Implemented array garbage collection routines to support additions, deletions, and changes to knowledge base (advice, premises, relations, functions, domains, etc.)

(g) Storage of premise deductive interactions and dependencies in a connection graph.

(h) Storage of conceptual associations among relations, domains, functions, and premises in a concept graph.

(i) Storage of deductive subproblems in a problem graph that makes extensive use of structive sharing to eliminate duplicate nodes.

(j) Construction of over three hundred premises representing bibliographic, kinship, naval ship, and shipping/receiving data base applications.

(k) Extensive testing and checkout of DADM prototype with premises, advice rules, and associated data bases and compute functions.

(l) Implemented techniques to find the shortest-most plausible inference plans first.

(m) Implemented "Try-Harder" feature to grow deductive search space upon user request.

(n) Implemented techniques to effectively deal with incomplete information in queries (e.g., missing arguments), in plans (e.g., missing support for subproblems), and in answers (e.g., missing facts in data base).

(o) Implemented techniques to deductively decompose query problems into deduce, search, and compute subproblems and order subproblems for efficient solution.

(p) Implemented a deductive apparatus that is expressionally and derivationally complete. This assures that all answers to a users query may be found.

(q) Implemented a global planning strategy to quickly zero in on relevant subsets of premises to support user queries.

(r) Implemented a semantic advice (i.e., meta rule) file that automatically invokes premise and relation selection strategies as necessary to enhance system performance.

(s) Implemented forward, backward, and middle term chaining techniques that are automatically activated as appropriate.

(t) Implemented techniques to identify logical properties of binary relations and assign special purpose deduction routines to avoid the "recursive premise" problem.

(u) Wrote "DADM Function Description" (TM-6035, Philip Klahr, March 1978) that briefly describes the LISP functions comprising DADM.

(v) Wrote "Alternative Architectures for Deductively Augmented Data Management Systems" (TM-6005, Charles Kellogg, December 1977) that describes a migratable module architecture to support various realizations of user centered, deduction centered, and data centered versions of DADM.

(w) Published four papers (references 3, 4, 5, and 6) on our research results.

5. FUTURE PLANS AND RECOMMENDATIONS

Over the past two and one-half years DADM has grown into a robust developmental prototype that demonstrates considerable utility as an on-line decision aid, as a supporter of high level user views, and as a data analysis and evaluation aid.

The next step seems clear; to interface DADM with one or several backend data base systems and one or several user-oriented frontend language processors. Once this is done DADM can be moved into test bed environments in which its capabilities can be throughly evaluated and feedback can be obtained from actual users about necessary and desirable improvements.

In addition, we believe a continuing research effort should support investigations in the following areas:

(a) Use of richer, higher level forms of semantic advice.

(b) Further investigation of recursive premises and ways of avoiding them through use of higher order logical constructs and abstraction mechanisms such as abstract data types.

(c) Investigation of techniques to discriminate between productive and non-productive deductive paths and plans. This information could then be stored for later use in avoiding non-productive paths and following productive paths.

(d) Investigate the use of DADM to support semantic integrity checking and the application of data security constraints.

(e) Investigate use of DADM in distributed data base environments as an intelligent planner, controller, and generator of data base access strategies.

(f) Develop additional special features to support future knowledge/data base administrators (such as semantic tuning to specific applications).

(g) Investigate improved user displays and user interfaces.

(h) Investigate advantage of converting DADM programs to another high order language (PASCAL, C, etc.).

APPENDIX A - REFERENCES

1. Green, C.C., Theorem Proving by Resolution as a Basis for Question Answering Systems". In Machine Intelligence 4, Meltzer, B. and Michie, D. (Eds.), Edinburgh University Press, Edinburgh, 1969, 183-205.

2. Kellogg, C. H., Burger, J., Diller, T., and Fogt, T. The CONVERSE natural language data management system: current status and plans. Proceedings of the Symposium on Information Storage and Retrieval, ACM, New York, 1971, 33-46.

3. Kellogg, C., Klahr, P., and Travis, L., A Deductive Capability for Data Management. In Systems for Large Data Bases, Lockemann, P.C. and Neuhold, E.J. (Eds.), North Holland, Amsterdam, 1976, 181-196.

4. Kellogg, C., Klahr, P., and Travis L., "Deductive Methods for Large Data Bases." Proceedings of the Fifth International Joint Conference on Artifical Intelligence, MIT, Cambridge, 1977, 203-209.

5. Kellogg, C., Klahr, P., and Travis L., "Deductive Planning and Pathfinding for Relational Data Base." In Logic and Data Bases, Gallaire H., and Minker, J. (Eds), Plenum, New York, 1978.

6. Klahr, P., "Planning Techniques for Rule Selection in Deductive Question-Answering." In Pattern-Directed Inference Systems, Waterman, D. and Hayes-Roth, F. (Eds.), Academic Press, New York, 1978.

7. Kowalski, R., "A Proof Procedure Using Connection Graphs. Journal of the ACM, 22, 1975, 572-595.

8. McSkimin, J. R., "The Use of Semantic Information in Deductive Question-Answer systems. TR-465, University of Maryland, College Park, 1976.

9. Minker, J., Fishman, D. H., and McSkimin, J.R., "The Q* Algorithm--a Search Strategy for a Deductive Question-Answering System." Artificial Intelligence, 4, 1973, 225-243.

10. Robinson, J. A., "A Machine-Oriented Logic Based on the Resolution Principle." Journal of the ACM, 12, 1965, 23-41.

11. Schwarcz, R. M., Burger, J. F., and Simmons, R. F., "A Deductive Question-Answerer for Natural Language Inference", Communications of the ACM, 13, 1970, 167-183.

12. Shostak, R. F., "Refutation Graphs. Artificial Intelligence, 7, 1976, 52-64.

13. Sickel, s., "A Search Technique for Clause Interconnectivity Graphs." IEEE Transaction on Computers, C-25, 1976, 823-835.

14. Simmons, R.F., "Answering English Questions by Computer: A Survey." Communications of the ACM, 13, 1965, 53-69.

15. Simmons, R.F., "Natural Language Question-Answering Systems: 1969." Communications of the ACM, 13, 1970, 15-30.

16. Shortliffe, E.H., Computer-Based Medical Consultations: MYCIN. American Elsevier, New York, 1976.

17. Travis, L., Kellogg, C., and Klahr, P., "Inferential Question-Answering Extending CONVERSE." SP-2697, System Development Corporation, Santa Monica, California, 1973.

18. Winograd, T., "Understanding Natural Language." Academic Press, New York, 1972.

19. Zadeh, L. A. Fuzzy Sets. Information and Control, 8, 1965, 338-353.

PART II

TABLE OF CONTENTS

1. REVIEW OF WORK ACCOMPLISHED

During the contractual period from October 1977 through 31 January 1979, several EUFID tasks were accomplished. The original proposal called for the following EUFID tasks:

(1) Task 1: Installation of EUFID with the METRO application at the customer site;

(2) Construction of the conceptual and semantic tables for the customer's Prototype I application;

(3) Task 3: Study of phonetic spelling correctors; and

(4) Task 4: Study of Negation.

1.1 TASK 1: INSTALLATION OF EUFID WITH THE METRO APPLICATION AT THE CUSTOMER SITE

EUFID was installed at the customer's site in December 1978. The system installed was the first demonstrable version of EUFID and a later version is planned for installation in March 1979. During installation we discovered that the EUFID system resulted in the customer's UNIX exhausting its swap space. This was not a EUFID-INGRES problem but rather a problem caused by the under-allocation of disk resources under UNIX. The swap problem occurs when there are so many processes running on UNIX that there is insufficient disk room in a pre-allocated disk segment to store core images of swapped-out processes. The customer is currently exploring a solution to the problem which would involve increasing the size of the swap space.

1.2 TASK 2: CONSTRUCTION OF THE CONCEPTUAL AND SEMANTIC TABLES FOR THE CUSTOMER'S PROTOTYPE I APPLICATION

We aided the customer with the construction of the conceptual tables (i.e., data base tables). The set of representative queries and application description necessary in order to build the semantic dictionary were not defined by the customer and delivered to us until November 1978. There

were neither funds nor time left to complete the building of the Prototype I sementic dictionary at that time. It will be possible to perform this work at a later date.

1.3 TASK 3: A STUDY OF PHONETIC SPELLING CORRECTORS

A study of phonetic spelling correctors was performed. The document TM-5711/400/00, "A Phonetic Spelling Corrector for EUFID," was delivered in October 1978. The study examined the character-based INTERLISP spelling corrector, and the SOUNDEX and IBM ALPHA phonetic name encoding procedures. For EUFID applications in which proper names occur frequently, it is suggested that the IBM ALPHA algorithm be implemented with modifications representing additional systematic orthographic-to-pronunciation rules.

1.4 TASK 4: STUDY OF NEGATION

A study of negation was performed. The document SP-3996, "Enabling EUFID to Handle Negative Expressions," was delivered in October. As a first step toward selecting negative expressions to be added to the EUFID vocabulary, a list of expressions categorized as negative in the linguistic literature was compiled. We then used as many as possible of these expressions in formulating questions for METRO, one of the applications being used as a testbed for EUFID. Additional negative expressions were found by constructing paraphrases of those questions. EUFID staff members rated each of the questions and its paraphrase(s) according to the likelihood that a question of that form would be asked by a EUFID user. On the basis of these ratings, 21 of the highest-scoring negative expressions were selected for detailed study. As a result of this detailed study, it was proposed that negative expressions be added to EUFID in the following order:

Stage 1: The pure negatives: "non-", "not", "n't", "un-", "outside'.

Stage 2: Negative qualifiers: "fewer", "the fewest", "the least", "less", "never", "no", "not...anything". (A few of these such as "the least" and "less" are already being handled in the prototype version of EUFID.)

Stage 3: Negative conjunctions: "but not", "neither...nor", "not (...) both...and", "not(...) either...or", "not only...but also".

Stage 4: Restrictives: "only".

Stage 5: Except/other expressions: "apart from", "besides", "except".

2. THE CURRENT EUFID SYSTEM

The prototype EUFID system is currently undergoing extensive system checkout at SDC in Santa Monica. A well tested version is expected to be installed at two customer sites by March 1979. A paper on the EUFID system was presented at the Fourth International Conference on Very Large Data Bases [1].

2.1 OVERVIEW

EUFID is a man-machine interface system that will permit users of data management systems to communicate with those systems in natural language. At the same time, EUFID will act as a security screen to prevent unauthorized users from having access to particular fields in a data base. The specific objective is to build a system that will be practical, efficient, and widely usable in existing, real-world applications. The approach is to model the restricted set of linguistic structures and functions required for each application, rather than the manifold linguistic properties of natural language per se. This allows the system to be powerful enough to efficiently process English queries against specific data bases without attempting to understand forms of English that have little or no function in the contexts of those data bases.

Why is a natural language interface necessary? Data bases are growing in number, size, and complexity. Data management systems have been built to accommodate this growth, but there is no standardization among DMSs as to language and the functions they perform. The number of casual users who need to retrieve information from data bases is also

growing. Currently, the approach of attempting to train groups of casual users to learn how to use one or more DMS languages and to become familiar with the data bases they need to access has not been successful. Moreover, as the cost of computer terminals continues to drop, it becomes increasingly more practical to make terminals available to casual users and enable them to easily retrieve their own information--an advantage they cannot now avail themselves of. A natural language interface will allow the casual user this easy access to data base information.

EUFID has been designed to be interactive and "friendly" to the user. We expect the typical EUFID user to have little experience with computers, data management systems, or even the organization of the data base from which he needs answers. It is necessary, however, that the user be competent in his application area, that the application area be well defined and bounded, and that all users competent in the application area share a common language of communication.

EUFID is a table-driven system. To support a new application in EUFID, we implement a new set of tables. The tables contain two different descriptions, or representations, of the application. One is that of the data base--its structure and semantics. The other is that of the syntax and semantics of the language a competent user uses to ask questions about the application.

EUFID has been designed to: (1) be a friendly interface to the casual user; (2) achieve a separation of the application into a user's view and a data base view; (3) handle the interfacing to both network and relational DMSs; (4) be application independent; (5) be portable; and (6) be able to reside on a minicomputer. The advantage to having a natural language interface reside in a minicomputer front-end machine is a practical one in that such an interface does not add to the usual overloaded conditions of the large frame computers.

There are two main groupings of EUFID modules: The table-building modules and the question-answering system. The table-building modules include the Application Definition Module, which builds the three main tables, and the Concept Graph Editor, which supports the editing of these tables for security reasons. These modules are used by the Data Base Administrator (DBA) or the Application Expert (AE) to build or edit the application-specific tables. All changes to the tables must be made through these modules. For integrity and security reasons, none of the EUFID users have direct access to these tables. The EUFID question-answering system reads the system tables but cannot alter them. Throughout this description we refer to the table containing information about the data bas as the "data base table"; the table containing the users' views of the application as "the semantic dictionary", and the table that maps the semantic dictionary into the data base as the "mapping table."

The EUFID system supports three types of interactive activity: question answering, synonym editing, and provision of help. Question answering is the main activity. When the EUFID user types a question on his terminal, the EUFID Controller reads it and forwards it to the Analyzer. The Analyzer interprets the question and produces a semantic representation of it. The mapper maps the semantic representation into a data base representation and generates an intermediate language (IL) representation. The Translator translates the IL into formal query-language statements for the specific data management system (DMS) and submits the query statements to the DMS. The DMS processes the query statements, accesses the data base, and sends the answer back to the EUFID user. The Analyzer and the Application Definition Module are the same for all DMSs and applications; a separate Translator is needed for each separate DMS. The effort required to build a Translator is directly proportional to the complexity of the DMS language used. A specialized set of tables is required for each application, and, again, the effort involved in building the application tables is related to the complexity of the application.

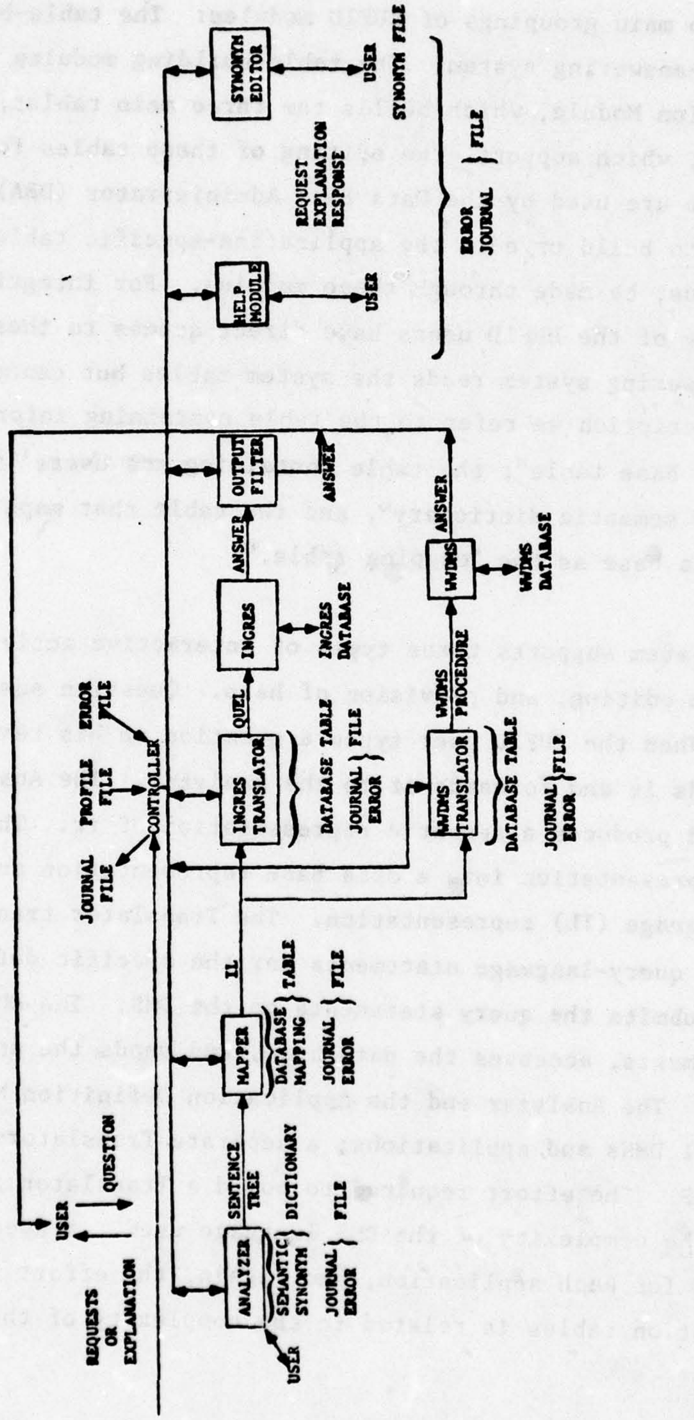


Figure 1. EUFID Operation and Control

The second type of user activity, synonym editing, is supported by the Synonym Editor Module. Each user of each application is allowed to create his own synonym dictionary to enable him to redefine words in the semantic dictionary. When the Analyzer is invoked it searches the synonym dictionary for words from the input question before it tries to find them in the semantic dictionary.

A third type of user activity is supported by the Help Module. When the user requests help, this module explains how EUFID operates and what it is capable of doing and not doing; it also provides a detailed explanation of the types of responses EUFID makes when it has been unsuccessful or has encountered some sort of failure. The Help Module is essentially an on-line users' manual.

Each EUFID user session generates a separate journal file. The journal consists of all interactions between the user and the EUFID.

2.2 STRUCTURE AND USE OF THE EUFID TABLES

Building a EUFID Application requires a negotiation process between an application expert and a EUFID consultant. This process is extremely important, and we are developing techniques for handling it [2]. The first step in the negotiation is for the application expert to prepare a description of the application and to collect a representative set of questions that competent users ask of the application. The next step is for the EUFID consultant to extract from the set of questions and the application description, a pictorial representation of the application world; this is a free-hand graph that shows the entities in the application and how they connect to each other by means of verbs, prepositions, etc. The EUFID consultant and the application expert then review the pictorial representation to ensure that it contains as complete semantic information as possible for eventual insertion into the application tables. When an adequate pictorial representation has been agreed upon, the data base is examined to ensure that all structures in the pictorial representation can be mapped into data base fields. When there are structures that do not map, then either new data fields need to be added to the data base or the application needs to be redefined to exclude

these semantic structures. This procedure also brings to light data base fields that have not been represented in the pictorial representation. The application expert may choose to build a representation for these fields in the pictorial representation or to ignore them if they are really not part of the application.

At the conclusion of the negotiations, the data base table, the semantic dictionary, and the mapping table are built.

Access to EUFID is controlled through a user profile table, which contains information about legal EUFID users, the applications and data bases they may access, and their table environment for a given application. The construction of this table is not part of the negotiation process, but is controlled by the Data Base Administrator.

The data-base table is composed of two parts: the CAN (canonical) and REL (relational) tables. They provide a common form for describing the logical structures of data bases implemented under different DMSs. All DMSs are designed to represent collections of data pertaining to entities and their attributes and the relationships between entities. A large variety of terms are used in existing DMSs to refer to these elements. The terms we are using are <u>group</u>, <u>aggregate</u>, <u>field</u>, <u>link</u>, and <u>domain</u>. In general, a group corresponds to an entity, an aggregate to a name for a set of fields, a field to an attribute, a link to a relationship, and a domain to fields having common values.

Each CAN table entry contains identifying information about a group, aggregate, or field in the data base, such as: its name; its identification as a group, aggregate or field; for aggregates and fields the immediate group or aggregate to which they belong; a unit code for fields whose values refer to unit

measures such as feet, miles, etc., the domain to which the values of a field belong (if appropriate); an upper/lower case flag; an alphanumeric flag; and an English name (or phrase) to be used as an output identifier for each field.

The REL table contains an entry for each group and lists every other group with which it has a primary or secondary link(s), and the linking field(s) in each group. For network data bases the link is the chain name for a chain that connects the master and detail records. If there are multiple chains between the master and detail, only one will be present in the REL table. The full set of chains are available to the Translator, and the choice as to which chain is applicable in the particular instance is made by the Translator. The REL table also contains a list of the fields that uniquely identify a group entry and a list of the fields that need to be included in the output answer whenever a question is asked about the group.

One of the purposes of the REL table is to define secondary or non-primary links between groups. An example of a secondary link is the date domain. When users ask questions such as "What warehouses were built after Ajax began shipping to Colonial?", the connection is made on the basis of date (i.e., the names of warehouses whose completion date is greater than the date when Ajax began shipping to Colonial). Secondary link information is semantic information about data base fields that is not easily elicited from the data base users. Most users, when asked how one data base group relates to another, are quick to mention the primary link relation but are not overtly aware of the secondary links until they need to use them to answer a question. The identification of fields belonging to the same domain, which is obtained during the negotiation process, furnishes this important linking information to the system.

Semantic Dictionary

The semantic dictionary is the most complicated table structure in the EUFID system. Words used to communicate about the application are defined

as to how they connect to each other in a generalized case grammar structure.

There are seven entry types in the semantic dictionary: (1) entities (e.g., nouns); (2) events (e.g., verbs); (3) functions; (4) parts of a phrase or idiom; (5) connectors (e.g., prepositions); (6) system words (e.g., conjuncts, auxiliaries, determiners); (7) anaphores (words that refer to previous words). All attributes are defined as entities. Entities that have no case structure beneath them are called 'primitive' entities. It might seem unusual to refer to the case structure of an entity, but in the semantic dictionary entities and events both have a similar case structure. Functions also have a case structure. The cases of functions are filled by their arguments.

The two main types of entries are entities and events. The orthographic spelling of the entry is followed by its type. (If an entry can be used as more than one type, then it is multiply defined.) A set of one or more senses is listed for each definition of the entry. Each sense has a node name to identify it, a pointer to a set of cases that define the sense, and the number of cases that need to be filled for this sense to be accepted (by the Analyzer in "understanding" a question) as the meaning of this sense of the entry. For each case, there is a set of one or more node names that can fill the case; an indication of whether the case is optional or obligatory; an indication of whether the case filler word occurs before or after the entry; a pointer to a set of acceptable connectors, any of which can connect the case filler word to the entry; an indication of whether or not this case filler word merges with the sense of the entry to determine its meaning; and a default indicator and default case fillers.

Figure 2 shows two examples of senses of the event "ship" (one active and one passive) and an example of the entity "shipment", which is recognized as a nominalized form of "ship". Notice that the cases have been arbitrarily labelled "CASE A", "CASE B", and "CASE C", rather than being

called the agentive case, the objective case, and the destination case. The case is whatever it is defined to be by its case filler, connector, etc., and by the sense for which it is a case. If we look at the active sense in detail, we can see that CASE A has the case filler of "shipping company", it is obligatory, and it must occur in a question logically before "ship". CASE B has the case filler of "part", it is optional, and it occurs logically after the word "ship". CASE C has the case filler "receiving company", it is obligatory, it must occur logically after the word "ship", and it requires the connector "to".

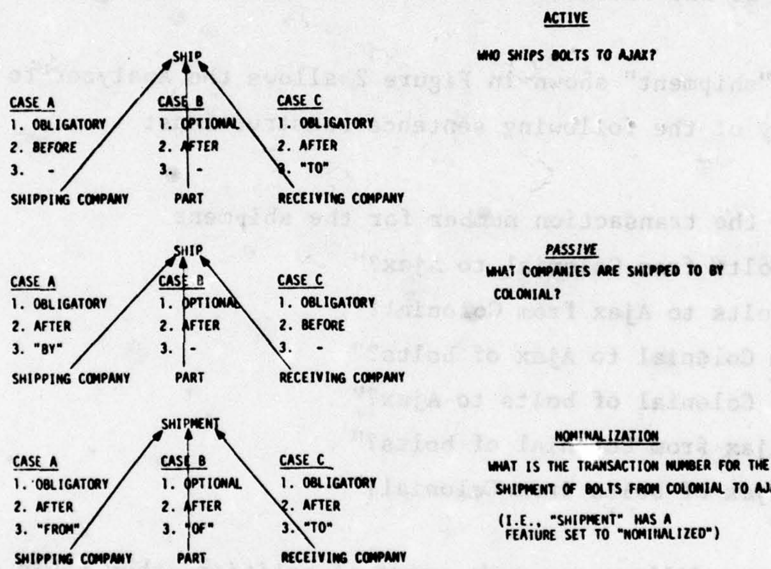


Figure 2. Example of Case Structure

If we compare CASE A of the active sense of "ship" with CASE A of the passive sense of "ship", we can see that they have the same case filler, "shipping company", and they are obligatory, but for the active CASE A, "shipping company" occurs before "ship" and takes no connector, while for the passive CASE A of "ship", "shipping company" occurs after "ship" and takes the connector "by". The same kind of comparison can be made for CASE B and CASE C and for the cases of the sense of "shipment". By lining the cases up the way they are presented in Figure 2, we could even label both the active and passive senses of "ship" and the sense of the nominalized form of "shipment" with the same node name.

The sense of "shipment" shown in Figure 2 allows the Analyzer to handle the meaning of any of the following sentence constructions:

"What is the transaction number for the shipment
of bolts from Colonial to Ajax?"
of bolts to Ajax from Colonial?"
from Colonial to Ajax of bolts?"
from Colonial of bolts to Ajax?"
to Ajax from Colonial of bolts?"
to Ajax of bolts from Colonial?"

Because the case fillers are node names of entities, they could each have a structure of cases hanging off of them. The question, "What is the transaction number for the shipment of number 3 bolts from Colonial in New York City, to Ajax in San Francisco?" would be analyzed similarly to the example in Figure 2.

Mapping Table

The mapping table is used to map the node names, or structures in the completed sentence tree, into data base field names. Each entry in the table has a node name followed by two parts. The first part describes the pattern of cases and their case fillers for that node name; the second part describes the production for each of the case fillers in the pattern part. A production may result in mapping a case filler to a data base field name, which is the desired final effect. When the sentence tree is several levels deep, it may be necessary to

map node names at lower levels of the tree to themselves or to synthesized variables. At higher levels, these node names will be cases on other node names and will eventually be mapped into data base field names. The reason for this complexity is that the data base field that the node name maps to may be dependent on the case that the node name fills for a higher-level node name. For instance, "company name" may map to a different data base field name if the question is "What companies are located in Los Angeles?" than it would if the question were "What companies ship to Los Angeles?" In the first question, "company" may map to a data base field name in a group containing information about companies in general, i.e., their addresses, phone numbers, presidents, etc. In the second question, "company" has the role of "shipping company" and may map to a data base field name of a group containing information about a shipping-receiving relation between companies.

2.3 EUFID QUESTION ANSWERING SYSTEM

Whenever EUFID is ready to accept an input from the user, it types "ready" on the user terminal. There are three words recognized as special words by EUFID: "help", "synonym", and "comment". "help" activates the help module, "synonym" activates the Synonym Editor, and "comment" results in EUFID asking the user to enter a comment into the system journal. If the user has not typed in any of the three special words, the Controller assumes the user has asked a question and it sends the question to the Analyzer.

Analyzer

The Analyzer is composed of two parts: the scanner and the Analyzer.

Scanner

The scanner begins by breaking up the user's input question into tokens and entering them into consecutive entries of the sentence list that are called word boxes. The Analyzer keeps the sentence list from previous questions (if there were any) to use in resolving anaphoric references that may occur in interpreting the current question. The scanner then looks up the definition for each token, first in the synonym dictionary and then in the semantic dictionary. When a definition is found, it is entered in the appropriate word box. If a definition cannot be found, then a morphological analysis [3] of the token is performed to split the token into a root and an ending. If this is

successful, the root is looked up in the dictionaries; if a definition for it is found, the definition is entered in the word box for the token along with inflectional information about the ending. If the morphological analysis is unsuccessful, the token is interpreted as a pattern, and the pattern is looked up in the semantic dictionary.

Pattern identification is a special feature of EUFID used for value inference. The semantic dictionary does not contain all the unique values present in the data base; it is designed to contain only those values, for particular domains, that form a fairly closed, small set (i.e., the values in the set will not change often). Other types of values, such as social security numbers, dates, transaction numbers, etc., which may be changing constantly, are inferred from their patterns. For example, if the question "What is the name and address of 123-45-6788?" were asked, the scanner would not find 123-45-6788 in either dictionary, nor would the morphology routine be able to split it into a root and ending. Instead, it would be interpreted as the pattern N(3)-N(2)-N(4), which would be looked up in the semantic dictionary and found to be a pattern for the node name "social security number." The definition for "social security number" would be entered into the word box for token "123-45-6788", so that a data base containing specific social security numbers can be successfully queried. The decisions as to which sets of values will be inferred and which will be entered into the dictionary are made during the negotiation process between the application expert and the EUFID Consultant. If a token pattern cannot be found in the dictionary, then the last attempt is to assume that a word was incorrectly spelled and apply the spelling corrector to it. It should be noted that the spelling corrector cannot be used for those applications in which value inference is applied to values having a completely alphabetic pattern (i.e., if value inference is applied to values of completely alphabetic patterns, then misspelled words will fit the alphabetic pattern and will not make it to the spelling corrector phase).

When the scanner has processed all tokens in the sentence list, it makes a roots list of all the trees--each tree pointing to a unique word box.

Analyzer

The purpose of the analysis is to find a way to connect all the trees in the roots list into a single tree.

The analysis is performed by processors, each of which handle connections between a specific group of entry types. The names of the different processors are: entity, event, function, anaphore, and conjunct.

The analysis is done by making left to right passes through the roots list. Regions are demarcated by right and left boundaries that are set by the various processors. The most important rule governing the analysis is that a tree may be connected only to a tree that is to its immediate left or right. Where a connection is made, the dominant tree becomes the tree top or father, and the other tree becomes the son. The connections are made on the basis of the case structure contained in the semantic dictionary.

For entity processing, the case connections are handled as described above, except in a situation where two adjacent entities can each fill a case on the other. When this situation arises the ambiguity is resolved using an algorithm taken from the conceptual processing work done at SDC [4].

The analysis is probably easier to understand if we demonstrate it by analyzing the simple question: "What companies in Chicago are shipping to Ajax?"

The output from the scanner would be a roots list of connected trees, each pointing to a word box as shown in Figure 3.

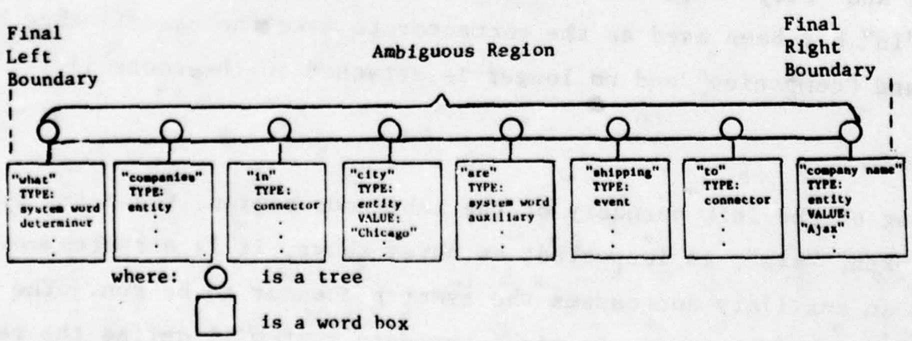


Figure 3.

A single region is defined by boundaries before the first tree and after the last tree. The region is ambiguous because it has not yet been processed.

Beginning at the left boundary, the definitional type of the first token "what" is inspected. It is a system word type that is a determiner and causes the entity processor to be run. The entity processor begins by finding a right boundary that will define the region within which it will process. In the example, the boundary is found at "are," which is a system word type that because it is an auxiliary, is processed by the event processor.

The entity processor then makes tree connections within the region "What companies in Chicago", based on the case structure defined in the word box definitions (See Figure 4).

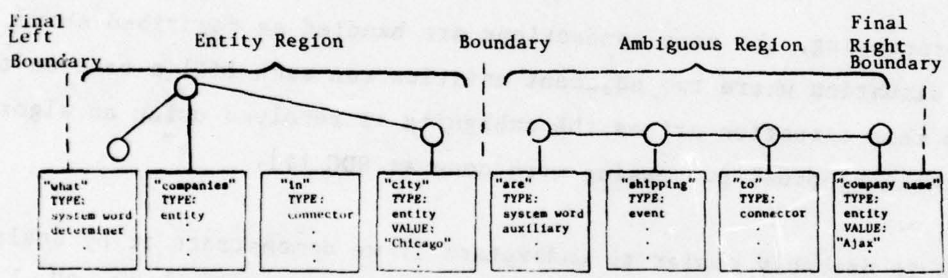


Figure 4.

The tree structure now shows "companies" with "what" connected to it through one case and "city" with the value of "Chicago" connected through another case. "In" has been used as the connector to make the case linkage between "city" and "companies" and no longer is attached to the roots list.

Beginning at the left boundary of the ambiguous region, the entry type of the first token, "are", is inspected; as noted above, it is a system word type that is an auxiliary and causes the event processor to be run. The event processor begins by finding a right boundary that will define the region within which it will process. This boundary is after the event "shipping" and before the connector "to".

The event processor subsumes the auxiliary "are" into the event "shipping", and "are" no longer is attached to the roots list. Part of the event processing involves looking across boundaries into unambiguous regions in order to determine if the event can be attached to a neighboring tree. The event processor looks to the left and finds that "companies" can fill a case of "shipping" and makes that connection. The roots list now looks like this.

The event processor continues processing to the left but finds only the final left boundary. It then tries processing to the right but is stopped at the boundary of the ambiguous region as shown in Figure 5.

Again, beginning at the left boundary of the region, the entry type of the first token, "to", is inspected. It is a connector type and causes the entity processor to be run. The entity processor runs and defines its entity region to be from "to" to the final right boundary. It is unable to make connections within its region, since it cannot connect the connector "to" to the entity "company name". There is a final right boundary after "company name", and detection of this causes the driver to examine the roots list. The roots list contains more than one tree, and processing begins again at the final left boundary.

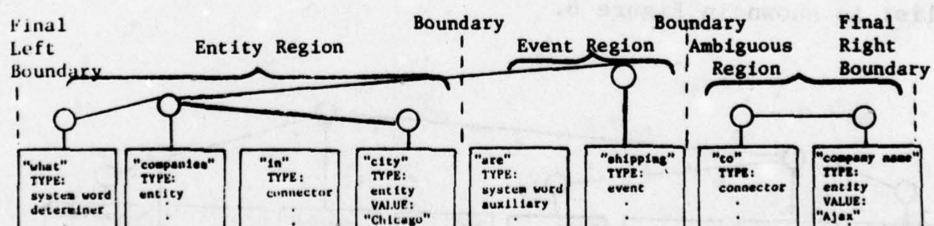


Figure 5.

The entry type of the first token, the tree top "shipping" is inspected. It is of type event and causes the event processor to run. The event processor looks to the left and finds only a final boundary, it looks to the right and finds an entity region. It tries to connect to the entity region but there is no case connection between "shipping" and "company name". The event processor stops. Beginning at the left boundary, the entry type of the first token "to" is inspected. It is a connector and causes the entity processor to run. The entity processor again cannot make a connection. Since the entity processor ends at the final right boundary, the driver is called. It inspects the roots list and detects that no connections were made during the previous pass and that there are two regions, one of which is an entity region. The entity processor is called, and it checks to see if an entity within the region is a name identifier for another entity. If so, it creates a new node that is the entity for which the entity name is a name identifier.

Processing again begins from the left. The event processor is called and again tries to make a connection to the right. This time it is successful, since "company" fills a case of "shipping", through the connector "to". The connection is made, the final right boundary is detected, and the driver examines the roots list and finds there is only one tree. The analysis is complete, and the final roots list is shown in Figure 6.

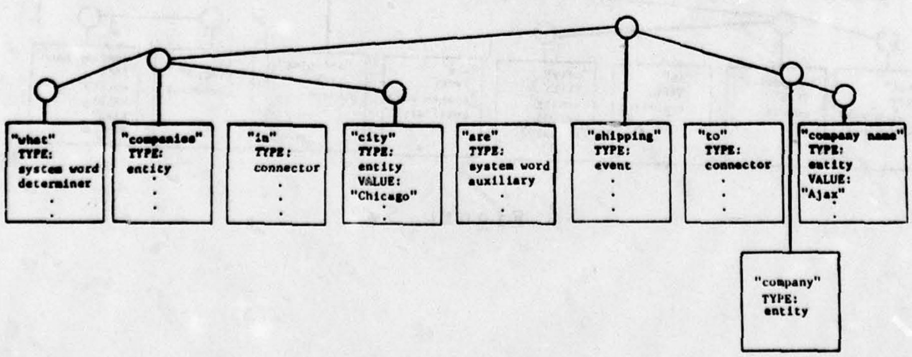


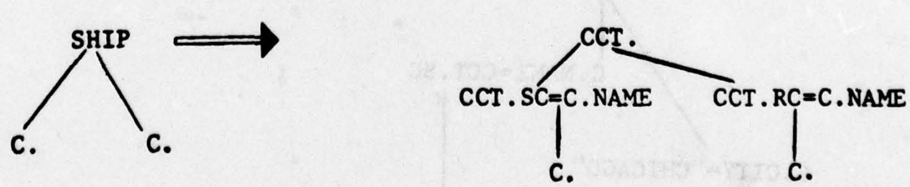
Figure 6.

Mapper

The purpose of the Mapper is to take the sentence tree output (or roots list) of the analysis expressed in the node names and to (1) restructure the tree, (2) redefine it through the mapping functions into a data base tree expressed in data base group and field names, and (3) build the intermediate language (IL) representation of the question by applying IL syntax rules to the data base tree.

The restructuring of the tree is mainly concerned with the logical distribution of conjuncts, which is necessary in order to produce an IL representation that is translatable into DMS query statements. For example, restructuring the sentence tree for the question "What companies ship perishables to Ajax and Colonial?" produces a new sentence tree that would have been produced if the question asked had been "What companies that ship perishables to Ajax, ship perishables to Colonial?" This needs to be done because "Ajax" and "Colonial" *are both values for the same case* filler, "company name." A company cannot ship, in the same shipping transaction, to two different companies.

Mapping functions are then applied to the node names on the tree to map the semantic dictionary structures into the data base fields, groups, and functions. It is through the mapping functions that the company cases of "ship" acquire the meaning "shipping company" or "receiving company". A graphical representation of a mapping function for "ship" is:



The mapping function for "company named X" is:

```
COMPANY           ───────►   C.
   |                          |
COMPANY NAME "X"             C.NAME=X
```

The data base group and field names refer to a relational data base where C. is a company relation in which C.NAME is the company name field and C.CITY is city field in which the company is located. CCT. is a company to company transaction relation where CCT.SC is the shipping company name and CCT.RC is the receiving company name.

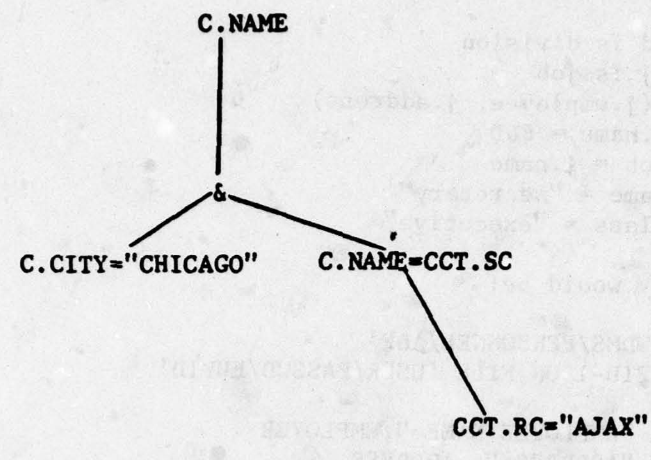
Below is a sentence tree of node names from the roots list output of the Analyzer for the question "What companies in Chicago are shipping to Ajax?

```
COMPANY
   \
    CITY
    "CHICAGO"
         \
          SHIP
            \
             COMPANY
               \
                COMPANY NAME
                  "AJAX"
```

After the mapping functions are applied, the same tree expressed in terms of data base groups and fields looks like this:

```
                C.
               / |
              /  C.NAME=CCT.SC
             /          |
C.CITY="CHICAGO"        |
                        |
                       CCT.
                          \
                           CCT.RC=C.NAME
                                 |
                                 C.
                                 |
                                 C.NAME="AJAX"
```

After output identifiers are added, logical "and" and "or" inserted, and unnecessary structures eliminated, the tree looks like this.



The IL syntax rules are applied to the tree to produce the statements:

```
RETRIEVE [C.NAME]
  WHERE (C.CITY="CHICAGO")
      AND (C.NAME=CCT.SC)
      AND (CCT.RC="AJAX")
```

The translator translates the IL representation of the user's question into a DMS query and submits that query to the DMS. Though all Translators employ the same basic structure and purpose, a different Translator is required for each DMS. The complexity of the Translator design and implementation is directly related to the complexity of the query language of the DMS.

We are currently producing Translators for two different DMSs: WWDMS (World Wide Data Management System), which runs under GCOS-TSS on the Honeywell H-6000 computer, and INGRES, which runs under UNIX on the PDP-11/70 computer. The query input language to INGRES is called QUEL.

A comparison of the complexity of the two Translators is shown by the difference in the DMS formal query statements for the question, "What are the names and addresses of the executive secretaries in R&D?" For the INGRES data base we assume two relations: division and job. For the WWDMS data base we assume a master record of division and a detail record of job.

The INGRES query would be:

```
range of d is division
range of j is job
retrieve (j.employee, j.address)
  where d.name = R&D
  and d.job = j.name
  and j.name = "secretary"
  and j.class = "executive"
```

The WWDMS query would be:

```
INVOKE 'WWDMS/PERSONNEL/ADF'
REPORT EUFID-1 ON FILE 'USER/PASSWD/EUFID'
  FOR TTY
Q1.  LINE "EMPLOYEE NAME=",EMPLOYEE
Q2.  LINE "ADDRESS=", ADDRESS
R1.  RETRIEVE E-DIVISION
       WHERE DNAME = "R&D"
     WHEN R1.
R2.  RETRIEVE E-JOB
       WHERE JNAME - "SECRETARY"
       AND CLASS = "EXECUTIVE"
     WHEN R2.
     PRINT Q1
     PRINT Q2
     END
```

The WWDMS query language is procedural and allows much of the expressive capability of higher-level programming languages. The INGRES language is basically non-procedural and does not allow for such things as report formatting.

Although the WWDMS Translator supports only a subset of the WWDMS query language, it is a much more complicated module than the INGRES Translator. One of its most complex features is the selection of an access path to the necessary data base fields.

In addition to the data definition for the I-D-S type data base, WWDMS supports a separate application definition file (ADF) that contains the names for the different access paths through the data base. Part of the application definition process for a WWDMS application involves defining a query ADF that enables the WWDMS Translator to select the optimal access

path for a query based upon the query's pattern of selector and qualifier record-types. The selector record-types are those that contain fields having values that the user wishes to see. The qualifier record-types are those for which the user has specified the field values.

The INGRES Translator begins its operation by passing the intermediate language through a lexical analyzer that identifies tokens as being specific types. The parser operates on the token list by rule. If a sequence of tokens fits a rule pattern, then a series of actions takes place. The purpose of these actions is to build a tree that structurally represents the QUEL output. When the QUEL tree is completed, the QUEL generator operates it to produce a QUEL query to send to INGRES. The rules and actions are built by using YACC (Yet Another Compiler Compiler). [5].

The WWDMS Translator uses the same lexical analyzer as the INGRES Translator to identify the intermediate language tokens. Its parser is also built by using YACC. It uses rules similar to those used by the INGRES Translator but takes different actions. All data base group and field references are passed onto an entry selection process that finds an optimal path solution for the pattern of selector and qualifier records for the particular query. The solution may involve nested retrieves or more than one independent retrieve. The query name(s) and other output from the parser are then brought together in the WWDMS procedure generator, which generates the statements, orders them properly, and writes them into a file to be compiled and run by WWDMS.

A question often asked is, "Why the elaborate process of building and providing the intermediate language only to have it immediately relabelled and interpreted by the Translator parsers?" The reason is that, because the Analyzer serves all DMSs, it can build output that is not translatable for a particular DMS. For example, the Analyzer could have "understood" the input question in terms of the user's semantics, but the intermediate language produced is an imbedded query. The answers from the inner query need to be saved and used as qualifiers for the outer query.

If a particular Translator cannot build the query structure to save the output of the imbedded query, it is not able to handle this type of situation and tells the user "EUFID can understand your question, but your data management system cannot handle it as one query. Please restate it by breaking it into multiple queries." An example of this type of situation is given by the question, "What employees earn more than John Doe?" Some DMSs require the question, "What is John Doe's salary?" followed by the DMS answer, "John Doe Salary $XXX", followed by a second question, "What employee's earn more than $XXX?"

In the EUFID system, each Translator assumes the responsibility of defining that subset of intermediate language it can translate into its DMS query statements. If the responsibility were not placed with the Translator, then the Analyzer could not be DMS independent.

3. FUTURE PLANS

EUFID is currently running under UNIX on the PDP-11/70 computer as an interface to INGRES, a relational data management system. The application is METRO; the relational data base contains shipping-freight transaction information.

EUFID is also currently running under UNIX on the PDP-11/70 computer as an interface to the World Wide Data Management System (WWDMS) which resides on a Honeywell H-6000 computer. The application is AIREP; the network-type data base contains information about software failure reports. EUFID accepts questions from the user and produces WWDMS queries that are then sent to WWDMS on the H-6000 computer for processing.

All EUFID components are written in RATFOR (a preprocessor for FORTRAN that allows some degree of structured programming) except for the Controller and the INGRES and WWDMS Translators which are written in C-language.

One of the most important tasks required in the immediate future is to perform EUFID operational testing. This task will require implementation of EUFID on additional operational data bases that may be accessed by WWDMS, INGRES or different data management systems. Operational testing will require the enlistment of the EUFID users in the evaluation of EUFID performance and use. Tuning of the Help module, Semantic Refusal and Guidance, use of the Synonym Editor and other features, will be done as a result of user evaluation.

Another top priority task in the future is to rewrite the Analyzer and Mapper in C-language. The C-compiler not only produces more optional code than the FORTRAN compiler but maintains separate data and instruction space and produces reentrant modules. Currently the Analyzer and Mapper reside in 6 separate modules because of space restrictions. The space restrictions of the Analyzer impose constraints on the length and type of questions that can be asked. For instance, the Analyzer is capable of handling compound verbs and relative clauses such as in questions like "What companies have been shipping light freight to warehouses in Lakeland?" or "What companies that ship light freight to Supreme ship perishable freight to Discount?" However these questions cannot currently be handled because the definitions for the words (i.e., the semantic net) is too large for the module size of the Analyzer. The rewriting in C-language would allow us to demonstrate the full capability of the Analyzer and the richness of the semantic dictionary (i.e., semantic net) and the mapping functions.

Another high priority task is in the table building area. In order to efficiently and effectively interface EUFID to new applications, work has to be done to (1) develop a more vigorous methodology for conducting the negotiations and (2) design and implement more automated ways of getting the acquired knowledge into the application tables. Currently, the knowledge acquired through negotiation is organized by hand, filled out on detailed forms, and keyed into a file from which the tables are built. What is needed is an intelligent module that can interact with the

application expert to help organize the knowledge, ask questions about the relations between entities, events and data base fields, and to the bookkeeping tasks involved in building semantic dictionary entries and mapping table functions.

We are also studying ways to bring the EUFID and DADM (Deductively Augmented Data Management, refer to final report) systems together. The EUFID Analyzer and IL would have to be expanded to handle modals and conditionals and DADM would read IL and produce IL for data base query. A more complicated version of the joining would be a EUFID-DADM front-end machine with multiple Translators able to communicate eith several DMSs on a single system or distributed over a network of computers. There is no inherent reason why an application needs to be confined to data fields in a single data base; the EUFID-DADM system could conceivably distribute a user's questions into a number of different queries to separate DMSs accessing various data bases and combine the multiple answers back for the user into an organized result.

In the future, we also plan to expand the Analyzer to handle negation and ellipses over multiple questions and improve the handling of anaphores. We would also like to study the problems of: temporal and spatial concepts; accepting answers back from the DMS and resturcturing them for the user (perhaps even in special ways such as graphs); helping the user to browse through his data base via the semantic dictionary; and handling multiple sentence questions.

APPENDIX A - REFERENCES

1. Kameny, I., Weiner, J., Crilley, M., Burger, J., Gates, R., Brill, D. "EUFID: The End User Friendly Interface to Data Management Systems," Fourth International Conference on Very Large Data Bases, West Berlin, September, 1978.

2. Weiner, J. L. "Deriving Data Base Specifications from User Queries", presented at the Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 25-27, Berkeley, California.

3. Winograd, T., "An A.I. Approach to English Morphemic Analysis", MIT AI Lab, Memo No. 241, February 1971.

4. Burger, J., Leal, A., Shoshani, A. "Semantic Based Parsing and a Natural-Language Interface for Interactive Data Management," AJCL Microfiche 32, 1975, 58-71.

5. Johnson, S. C. "YACC: Yet Another Compiler-Compiler," Bell Laboratories, Murray Hill, New Jersey 07974.

DISTRIBUTION

Defense Documentation Center Cameron Station Alexandria, VA 22314	12 copies
Office of Naval Research Information Systems Program Code 437 Arlington, VA 22217	2 copies
Office of Naval Research Code 102IP Arlington, VA 22217	6 copies
Office of Naval Research Code 200 Arlington, VA 22217	1 copy
Office of Naval Research Code 455 Arlington, VA 22217	1 copy
Office of Naval Research Code 458 Arlington, VA 22217	1 copy
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, MA 02210	1 copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, IL 60605	1 copy

DISTRIBUTION (cont'd)

Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, CA 91106	1 copy
New York Area Office 715 Broadway - 5th Floor New York, NY 10003	1 copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1 copy
Naval Electronics Laboratory Center Advanced Software Technology Division Code 5200 San Diego, CA 92152	1 copy
Mr. E. H. Gleissner Naval Ship Research & Development Center Computation and Mathematics Department Bethesda, MD 20084	1 copy
Captain Grace M. Hopper NAICOM/MIS Planning Branch (OP-916D) Office of Chief of Naval Operations Washington, D.C. 20350	1 copy
Mr. Kin B. Thompson Technical Director Information Systems Division (OP-91T) Office of Chief of Naval Operations Washington, D.C. 20350	1 copy
Advanced Research Projects Agency Information Processing Techniques 1400 Wilson Boulevard Arlington, VA 22209	1 copy